Vom Fachbereich für Mathematik und Informatik
der Technischen Universität Braunschweig
genehmigte Dissertation
zur Erlangung des Grades eines
Doktor-Ingenieurs (Dr.-Ing.)

Gordon Müller

Object Hierarchies for Efficient Rendering

Tag der mündlichen Prüfung: 24.05.2004

1. Referent: Prof. Dr. Dieter W. Fellner
2. Referent: Prof. Dr. Wolfgang Straßer
eingereicht am: 20.01.2003

Abstract

The efficient synthesis of computer generated images for large three-dimensional models is one of the major challenges of computer graphics research. Despite the rapid growth of available computational power of todays microprocessors and highly optimized graphics chips used for efficient rendering of graphical primitives, this growth was even outbeaten by the sheer increase of complexity of three-dimensional data sets to be visualized.

This growth of complexity made it necessary to invent acceleration schemes for different types of rendering algorithms. A common basic idea is to build spatial data structures that allow for efficient spatial retrieval of primitives in the rendering process. This retrieval may be used to detect visible objects efficiently or to detect invisible objects prior before sending them through the graphics pipeline, such that expensive computations like *shading* or *rasterization* may be omitted.

This thesis examines different acceleration schemes in detail and analyses related data structures for their utility within several rendering algorithms. We focus on acceleration schemes that make use of *object hierarchies*, i.e. scene geometry that is managed hierarchically.

We also present a novel technique for automated generation of object hierarchies. The essential idea of our algorithm is to build the hierarchy such that the average rendering costs when traversing the scene hierarchy are minimized. This technique applies successfully in the fields of photorealisitic visualization using *ray tracing* and *radiosity*. First, we achieve a significant improvement of run-time efficiency by accelerating the visibility tests. Also, we demonstrate that the generated object hierarchies may ideally be used to approximate the computation of light exchange between complex distant scene objects for *radiosity with clustering*. Only the use of objects clusters made it feasible to run an radiosity computation on huge data sets.

This thesis also covers the area of real-time rendering, especially with a deeper look at *occlusion culling* algorithms, which have attracted much attention in the recent past. Occlusion culling schemes try to detect occluded scene geometry early to remove these objects from the graphics pipeline. We describe the usefulness of object hierarchies within the occlusion culling process and compare different hierarchy generation schemes. By the utilization of our new hierarchy generation scheme we could achieve a significant increase in frame rate when rendering complex scenes.

Finally, we present a system for interactive visualization of huge data sets by combining occlusion culling and ray casting. This system increases the frame rate further compared to classical occlusion culling. Additionally, our method has only a minimal overhead for scenes with almost no occlusion. Occlusion tests are automatically activated and de-activated when needed.

Zusammenfassung

Die effiziente Bildsynthese für die computergenerierte Darstellung grosser dreidimensionaler Datenmodelle, ist nach wie vor eine der grössten Herausforderungen für die Forschung im Bereich der Computergrafik. Trotz des starken Wachstums der verfügbaren Rechenresourcen durch neue Mikroprozessoren und hochgradig optimierte Grafikchips, welche die effiziente Darstellung einfacher grafischer Grundobjekte beschleunigen, wurde doch dieses Wachstum noch durch den noch gewaltigeren Anstieg der Komplexität der 3D-Modelle geschlagen, die visualisiert werden sollen.

Dieses Problem hat es notwendig gemacht, Beschleuningungstechniken für unterschiedliche Renderingverfahren zu entwickeln. Eine Grundidee hierbei besteht darin, Datenstrukturen aufzubauen, die es dem Darstellungsprozess ermöglichen, effizient die relative räumliche Lage von Darstellungsobjekten abzuleiten. Sichtbare Objekte lassen sich so leichter identifizieren, bzw. nicht-sichtbare Objekte frühzeitig vor der Weiterverarbeitung eliminieren, um dadurch weitere aufwendige Berechnungen, wie etwa die *Beleuchtungsrechnung* oder die *Rasterisierung*, zu unterdrücken.

Diese Dissertation untersucht detailliert unterschiedliche Beschleunigungstechniken sowie die jeweils benötigten Datenstrukturen bezüglich ihrer Nützlichkeit in vielfältigen Darstellungsalgorithmen. Hierbei wird der Fokus auf Verfahren gelegt, die *Objekt-Hierarchien* ausnutzen, also eine Szenengeometrie hierarchisch verwalten.

In Rahmen dieser Arbeit wird ein neues Verfahren zur automatischen Hierarchiegenerierung präsentiert, wobei die Grundidee darin liegt, die Hierarchie derart aufzubauen, dass die erwarteten späteren Rendering-Kosten beim Traversieren der Szenengeometrie minimiert werden. Solche Hierarchien werden erfolgreich in dem Bereich der fotorealistischen Visualisierung mittels *Ray-Tracing* und *Radiosity* eingesetzt. Hierbei wird zum einen eine deutliche Verbesserung der Laufzeit durch eine Beschleunigung der Sichtbarkeitsberechnungen erzielt. Zum anderen wird für das Radiosity-Verfahren gezeigt, dass die erzeugten Objekt-Hierarchien ideal zur effizienten Berechnung des Lichtflusses zwischen entfernten, komplexen Szenenobjekten verwendet werden können (*Radiosity Clustering*). Hierdurch wurde die Verwendung sehr grosser Datensätze innerhalb der Radiosity-Berechnung überhaupt erst ermöglich.

Diese Arbeit geht ebenso auch auf den Bereich des Echtzeit-Renderings ein, dabei insbesondere auf *Occlusion Culling*-Algorithmen, denen in letzter Zeit viel Aufmerksamkeit zuteil wurde. Hierbei wird versucht, von anderen Objekten verdeckte Szenengeometrien frühzeitig zu entdecken, um sie aus der Graphikpipeline zu entfernen. Dabei wird die Nützlichkeit von Objekt-Hierarchien für den Prozess des Occlusion Cullings hervorgehoben und es werden unterschiedliche Generierungsalgorithmen für solche Hierarchien verglichen. Als Resultat konnte durch Einsatz eines neuen Verfahrens zu Hierarchieerzeugung eine deutliche Erhöhung der Bildgenerierungsrate bei komplexen Szenen erzielt werden.

Schliesslich wird ein System zur interaktiven Visualisierung grosser Szenenmodelle präsentiert, welches durch Kombination von Occlusion Culling und Strahlschnitttests wiederum eine Erhöhung der Bildgenerierungsrate im Vergleich zu klassischem Occlusion Culling erreicht. Es

wird zudem aufgezeigt, dass dieses Verfahren in Szenen mit geringer Verdeckung nahezu keinen Overhead beinhaltet. Die Occlusion-Tests werden hierbei genau dann automatisch aktiviert bzw. deaktiviert, wenn es notwendig ist.

Contents

List of Figures

List of Tables

Acknowledgements

First of all, I would like to thank my supervisor Dieter Fellner. He was both the initiator and the catalyst of this work. To cut a long story short, without his continuing support this thesis simply wouldn't exist.

Next, I wish to thank all members of the computer graphics groups at the Universities of Bonn and Braunschweig. They provided me fruitful discussions, broadband internet access, and a hailstorm of curious ideas about the ultimate rendering API. Especially, I want to thank my former collegue Stephan Schäfer. It was really fun!

I would also like to acknowledge the *Deutsche Forschungsgemeinschaft* (DFG) for funding this research.

Finally, I would like to thank my wife Angela Müller for everything else. She knows best what I mean.

CHAPTER 1

Introduction

1.1 Three-Dimensional Image Synthesis

The computer aided generation of images for both, photorealistic and real-time visualization, is the core of computer graphics research. This thesis is about rendering algorithms, i.e. methods that create computer generated images either as realistic still images or such that a viewer can interact with a virtual three-dimensional environment. Other interesting fields such as modeling, animation, spatial sound, user input, simulation etc. which are needed to build complex systems are beyond the scope of this thesis.

Especially the rise of more and more complex data sets has led to the need for new data structures and algorithms that process these data sets efficiently, e.g. allow the fast hierarchical retrieval of information. The generation of such data structures and the related information retrieval is the core of this thesis, but before we can describe these optimized methods we have to understand classic techniques used in the field of image synthesis to start from a solid base. First, we will present the rendering pipeline, which is the core of rendering systems, next we will present the basic ideas behind acceleration techniques used for hardware assisted rendering and global illumination algorithms such as *ray tracing* or *radiosity*.

Section 1.4 will conclude with a summary on the major contributions of this thesis and presents an overview on subsequent chapters.

1.2 The Graphics Rendering Pipeline

The *rendering pipeline* [Rog97] can be seen as an underlying tool for the rendering system. Its main scope is to generate two-dimensional images depending on a given scene database, a virtual camera, light sources, lighting models, textures, etc. This tool is typically subdivided into several stages. Figure 1.1 shows three conceptual stages which are always common to different implementations of a rendering pipeline.

Application: In this stage rendering primitives, like polygons or light sources, are set up under the control of an specific application. This stage is performed in software, since it is highly application specific.

Geometry: This stage handles per-polygon or per-vertex operations. It is mostly subdivided into stages such as model & view transformation, lighting, projection, clipping, and

screen mapping. Modern graphics cards implement this stage in hardware and even allow to re-program the per-vertex operations [LKM01].

Rasterizer: This stage handles per-pixel operations. Typically, in a scan-line conversion process projected and transformed vertices colors and texture coordinates from the geometry stage are used to convert two-dimensional vertices into pixels on the screen. Visibility is mostly resolved using the z-buffer-algorithm [Cat74]. Almost every graphics card implements this stage in hardware. Modern graphics cards also allow to combine several textures and colors in a single hardware cycle such that high-quality surface effects such as *bump mapping* [Bli78] can be done in real-time without rendering in multiple passes [DB97].

For efficient utilization of the rendering pipeline it is important to understand that pipeline stages may process input in parallel. New input to a specific stage may already be processed by the same stage when previous output completed (although the end stage of the total pipeline is not yet finished). This theoretically allows for massive speed-ups depending of the pipeline layout. Graphics developers always must have in mind not to block certain stages of the rendering pipeline: many tiny polygons with a projected size of only a few pixels will be fast to rasterize but a huge number will block the geometry stage. On the other hand a very large single polygon will stress the rasterizer while it will go very fast through the geometry stage. The slowest stage always determines the *rendering speed*, i.e. the update speed of the generated images. For this reason, the basic stages are mostly subdivided into sub-stages. Typically, it is up to the developer to care for good appropriate utilization within the application stage.

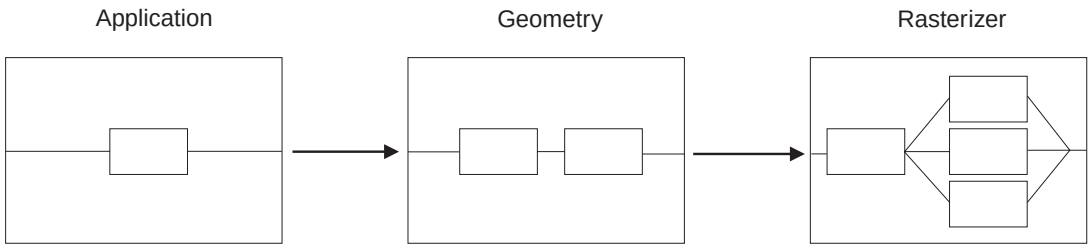


Figure 1.1 Rendering pipeline.

1.3 Illumination algorithms

1.3.1 Real-Time Rendering

Systems that allow for real-time rendering make simplifying assumptions on the shading steps within the geometry and rasterizing stage of the graphics pipeline. The illumination of a point or surface element only depends on the surface properties and the light sources which makes it rather easy to compute the illumination of a surface element. Mostly, *local* illumination algorithms found in current graphics hardware use variants of the Phong equation [Pho75] (Equation 1.1) to compute the intensity of a surface element to be illuminated. Assuming n light sources the illumination value I can be computed as

$$I = I_a k_a + \sum_{i=1}^{n} I_i (k_d \cos\theta_i + k_s \cos^c(2\alpha_i)) \tag{1.1}$$

where

- k_a, k_d, and k_s describe the ambient, diffuse and specular material properties respectively. In the past these values very often assumed to be constant after the geometry stage, but recent hardware with extended texture mapping capabilities [NVI01] allows to change these values efficiently even within the rasterizer stage on a per-pixel base.
- I_i describes the intensities of the light sources. It is important to understand that no check is performed if the surface elements are actually illuminated by this source or shadowed by other scene elements.
- θ_i describes the angle between the vector to the light source i and the surface normal at a given surface point. According to Lambert's law the amount of reflected light for diffuse surfaces is proportional to the cosine of this angle.
- The term $\cos^c(2\alpha_i)$ approximates the amount of light speculary reflected from a light source i into the direction of the eye. α_i is the angle between a perfect reflection vector on the surface according to light source i and the vector pointing to the eye point. c describes the sharpness of the reflection cone and is material specific. We can conclude from the formula that we get largest specular intensity values if the reflection vector and the eye vector coincide.
- I_a is the ambient term. It tries to approximate all illumination not directly received from light sources (indirect illumination).

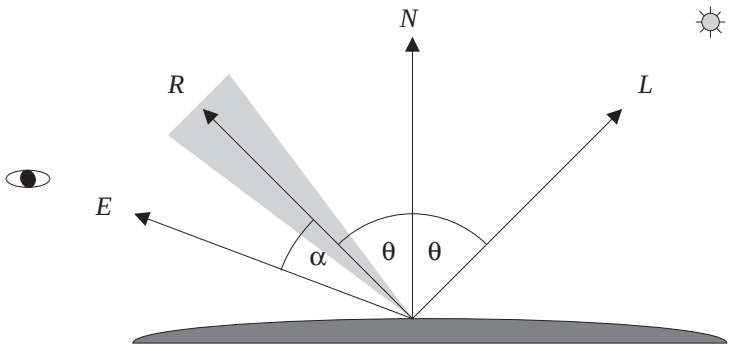


Figure 1.2 Vectors used in the Phong reflection model.

When rendering complex scenes containing hundreds of thousands of primitives, often only few of these primitives contribute to the final raster image. Primitives may be outside the field of view of an observer or may be occluded by other primtives (Figure 1.3). Although they do not contribute to they final image they either have to pass the geometry stage (primitive outside the field of view) or the rasterization stage (z-buffer tests) of the graphics pipeline.

Acceleration algorithms that detect invisible geometry are called *culling* algorithms. *View-frustum culling* detects objects outside the field of view and *occlusion culling* detects objects behind other scene objects before they are sent through the graphics pipeline. In Sections 5 and 6 we describe these algorithms in detail and present a new occlusion algorithm utilizing object hierarchies that structure the scene data base hierarchically, i.e. inner scene nodes hold sub-scenes as children.

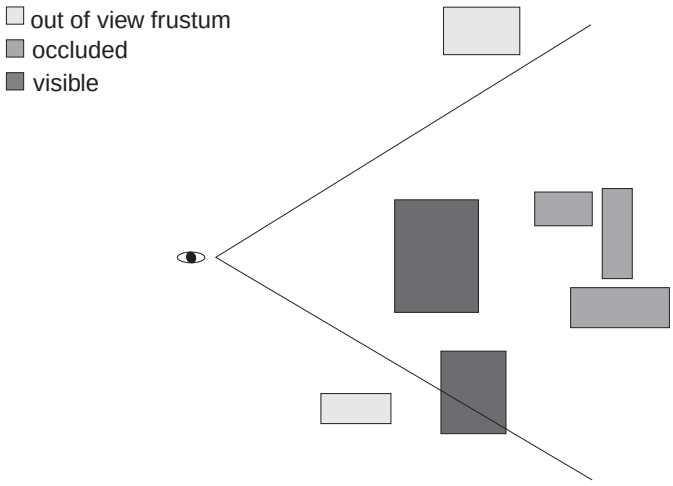


Figure 1.3 Visibility classification.

1.3.2 Ray Tracing

Ray tracing [Whi80] belongs to the family of *global* illumination techniques. Instead of using rough approximations of the indirect illumination like using a single ambient constant, ray tracing samples several directions from the point to be shaded to gather additional direct and indirect illumination contribution (Figure 1.4). In its simplest form a single ray sample is used along the direction of a perfect mirror to simulate the behaviour of specular reflective surfaces. Also, rays are traced to every light source to identify if a light source contributes to the illumination of this point. This process is repeated recursively until some predefined threshold on recursion depth or intensity contribution is reached. This scheme can easily be extended by also tracing a refracted ray according to the surface properties.

The quality of images generated by the ray tracing algorithm may easily extended by sending more than a single ray to sample the reflection cone of the surface point to be illuminated. Also, complex area light sources may be simulated by sampling the surface of the light source. Such techniques are summarized under the term *Distributed Ray Tracing* [RTL84], since rays are distributed according to some statistical distribution depending on surface properties. This idea is consequently generalized by techniques that utilize Monte Carlo integration [Kaj86] to take every possible way of light transport into consideration by sampling the whole hemisphere around a surface point.

For large scene geometries the visibility tests between light sources and surface points and traced reflection and refraction rays dominate the run-time of the ray tracing algorithm and its

variants. For this reason many acceleration schemes have been invented to find the intersection between a ray and a large set of scene objects. To minimize the number of objects that have to be tested indexing schemes were developed that group scene objects according to their spatial distribution [Gla89]. Chapter 3 presents a survey on ray acceleration techniques and compares them in detail.

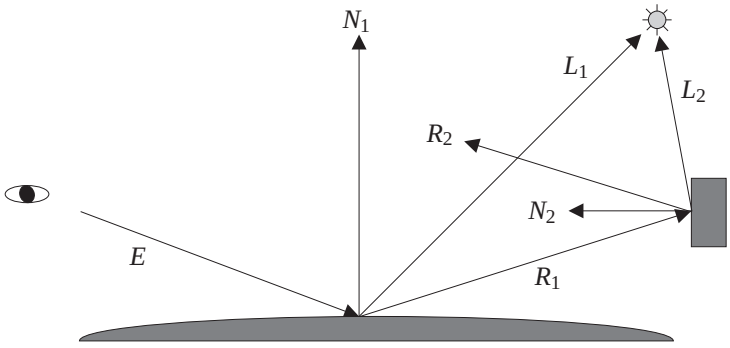


Figure 1.4 Recursively tracing illumination rays.

1.3.3 Radiosity

The *radiosity method* [GTGB84, NN85] is a different global illumination technique. It allows to compute the energy transfer between diffuse surfaces in closed systems by using results from the theory of radiative heat transfer. A complete discussion on radiosity algorithms is far beyond the scope of this thesis. In this section, we will only describe the basic foundation of the method and motivate where its computational complexity arises. For a complete overview on radiosity methods we refer to [Sch00].

Based on an energy equilibrium model, the total radiosity B_i of a surface patch F_i (which is the physical measure of the power radiated per unit area of a surface) equals the sum of its self-emissivity E_i plus the product of its reflectivity ρ_i (fraction of incoming energy that is reflected back) and the amount of energy arriving from all n patches (Equation 1.2). The amount of energy arriving from a single patch F_j can be written as the radiosity B_j times a geometrical term F_{ij} that describes the fraction of the total energy that arrives at F_i from the sending patch F_j. F_{ij} are called the *form factors* of the radiosity system.

$$B_i = E_i + \rho_i \sum_{j=1}^{N} F_{ij} B_j \tag{1.2}$$

Assuming constant radiosity over each surface patch the radiosity B_i may be computed by solving a system of n linear equations. Several publications cover the efficient solving of this system, e.g. [CCWG88] describes a progressive refinement method that only stores a single column of the linear equation system matrix and computes missing form factors dynamically only when needed.

The computation of form factor dominates the run-time of the radiosity method for every radiosity algorithm, since for every pair of patches a mutual visibility computation is needed

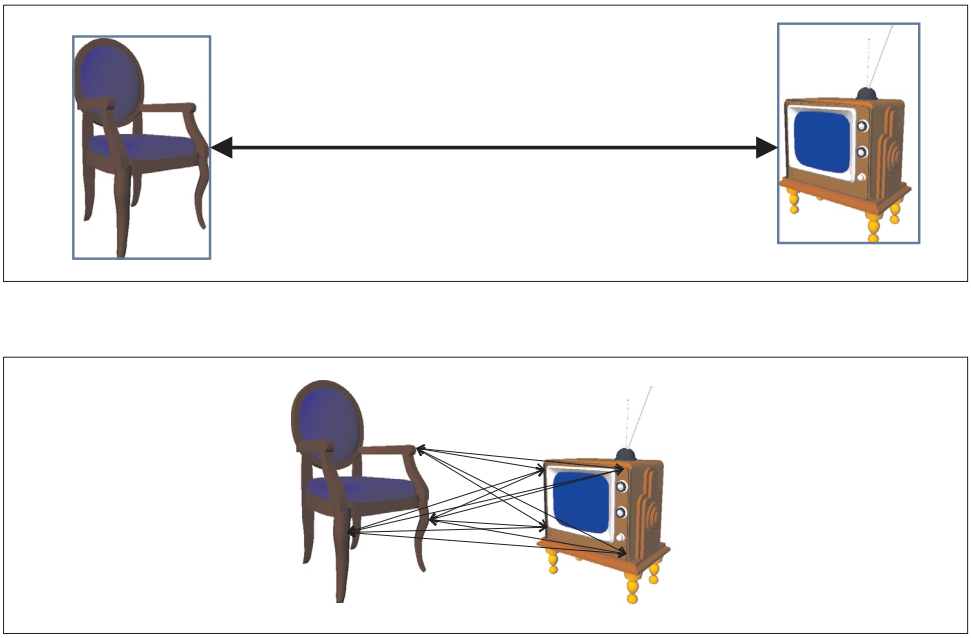


Figure 1.5 Radiosity Clustering: the computation of the energy transfer between distant complex geometry is sped-up by using object clusters instead of using a patch-to-patch model. (Figure by S. Schäfer [Sch00])

(which may be solved by tracing visibility rays between the patches). Further improvements on the radiosity method concentrated on the reduction of form factor computation by modeling the energy exchange at different resolution levels of the mesh (Hierarchical Radiosity [HSA91]).

The basic idea of the hierarchical radiosity algorithm was further extended by grouping blocks of patches hierarchically which led to radiosity clustering algorithms [Sil95, SAG94]. Figure 1.5 shows the main idea behind this acceleration technique: instead of computing the energy transfer between distant objects patch-by-patch, objects are grouped into clusters and energy transfer is computed between these clusters. In Chapter 4 we will present an algorithm for hierarchy generation that finds clusters of objects automatically which are appropriate for radiosity computations.

1.4 Thesis contribution and outline

This thesis contributes to the field of rendering acceleration by the use of object hierarchies. It also gives an overview on rendering algorithms such as ray tracing, radiosity and visibility culling. Chapter 2 presents an overview on available hierarchy generation techniques and shows how hierarchy information can be used within many rendering algorithms to speed them up.

Chapters 3 to 6 are the core of this thesis and present innovative techniques for rendering acceleration. These chapters consist of updated refereed conference and journal papers that have been slightly adapted to fit the style of this thesis. The basic idea behind these improvements over previously known techniques is the fact that the underlying object hierarchy is automatically generated by a minimization algorithm that tries to optimize the overall expected

rendering costs. The actual cost function varies among different rendering algorithms but the overall technique is mostly unchanged.

The contribution of Chapter 3 is two-fold. First, a new hierarchy generation scheme is developed that outperforms the standard technique from Goldsmith and Salmon [GS87] significantly. Possibly more important, a hybrid scheme is presented that combines different acceleration techniques (namely object hierarchies and space subdivision) with superior run-time behaviour than any of the single schemes. This work was originally published as [G. MÜLLER, D.W. FELLNER: Hybrid Scene Structuring with Application to Ray Tracing. *Proceedings of International Conference on Visual Computing (ICVC'99)*, pp. 19–26, Goa, India, February 1999].

In Chapter 4 we adapt the automatic hierarchy generation scheme to the field of radiosity [G. MÜLLER, S. SCHÄFER, D.W. FELLNER: Automatic Creation of Object Hierarchies for Radiosity Clustering. *Proceedings of Pacific Graphics '99 (Seventh Pacific Conference on Computer Graphics and Applications)*, IEEE Computer Society Press, Seoul, Korea, October 1999]. We demonstate how object clusters can automatically be found for improved radiosity algorithms such as hierarchical radiosity working with object clusters [Sil95, SAG94]. The clusters that are generated by the new algorithm fulfill many of the properties that have been previously reported as required [HDSD99]. This work was selected for the PG99 special edition of the Computer Graphics forum and re-published as [G. MÜLLER, S. SCHÄFER, D.W. FELLNER: Automatic Creation of Object Hierarchies for Radiosity Clustering. *Computer Graphics forum*, Vol. 19, No.4, number 4, pp. 213-221, December 2000].

Next, we apply our hierarchy generation to the field of real-time rendering in Chapter 5, more precisely to the field of hierarchical occlusion culling [GKM93]. We show how our adapted hierarchy generation scheme can be used to eliminate most of the occluded geometry of a huge scene geometry, thus yielding a performance boost compared to naive rendering [M. MEISSNER, D. BARTZ, T. HÜTTNER, G. MÜLLER, J. EINIGHAMMER: Generation of Decomposition Hierarchies for Efficient Occlusion Culling of Large Polygonal Models, *Vision, Modeling, and Visualization (VMV 2001)*, Stuttgart, November 2001]. A more detailed version of this chapter is also available as a technical report [M. MEISSNER, D. BARTZ, T. HÜTTNER, G. MÜLLER, J. EINIGHAMMER: Generation of Decomposition Hierarchies for Efficient Occlusion Culling of Large Polygonal Models, WSI/GRIS technical report, University of Tübingen, WSI-99-13, ISSN 0946-3852, June 1999].

Chapter 6 [G. MÜLLER, D.W. FELLNER: An Adaptive Hierarchical Occlusion Culling Algorithm for Interactive Large Model Visualization, *Computer Graphics forum*, 2001 (submitted)] presents many ideas in the field of hierarchical occlusion culling. We show how visibility queries may be answered by the apriori knowledge of partially visible and invisible geometry. This knowlegde is propagated into the object hierarchy which may lead to dramatic reduction of visibility queries. The chapter also presents a hybrid visibility determination scheme which combines standard z-buffered rendering with a ray casting approach to improve the rendering performance. In this framework, rays from the eyes are shot to sample the environment and collect visibility information for the z-buffer pass.

The thesis concludes with Chapter 7 which summarizes our work and presents an outlook.

CHAPTER 2

Object Hierarchies

2.1 Bounding Volumes

Bounding Volumes are probably the most valuable tool for the acceleration of rendering algorithms. A bounding volume of a geometrical object is defined as a (geometrically simple) body which embrasses the object completely (Figure 2.1). It must be guaranteed that every point within the geometrical object or on its surface lies within the body defined by the bounding volume. The empty space between the surface of the bounding volume and the bounded geometrical object is called the *void*-area.

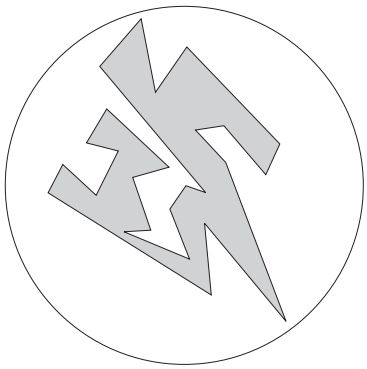


Figure 2.1 Spherical bounding volume embrassing a rather complex object.

2.1.1 Bounding Volumes and Visibility

Bounding volumes may help to speed-up rendering algorithms significantly by the acceleration of some underlying visibility tests. Figure 2.2a. shows an example where a rather complex object is outside the field-of-view of an observer. We can easily conclude that if the bounding volume of an object is not visible the object itself cannot be visible by definition. Algorithmically, we can draw our conclusion by a simple bounding volume / view-frustum test without looking at the (rather complex detail) of an underlying object. In Figure 2.2b. we can conclude from the knowlegde that the bounding volume is occluded by a closeby opaque barrier that the object must be invisible and further visibility tests may be omitted. Similary, in Figure 2.3 we can conclude that the ray does not hit the object since it does not hit the bounding volume.

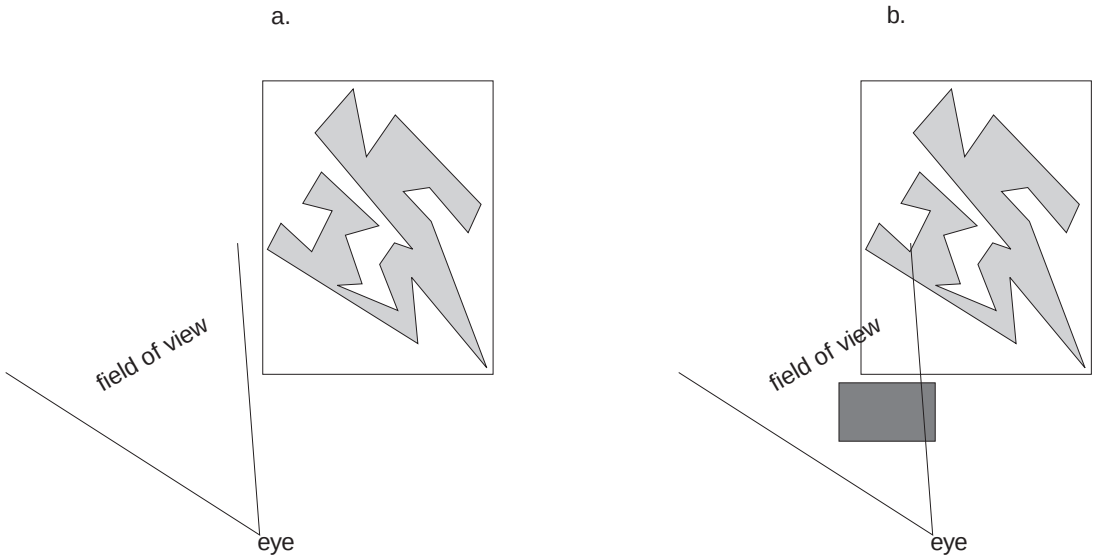


Figure 2.2 Non-visible bounding volumes relative to an observer, a. complex object outside the field-of-view, b. complex object occluded by a blocking object.

But there is also drawback involved in this test: the visibility test may come to the conclusion that the object is potentially visible because the bounding volume is (at least partially) visible. In this case the test did not help at all to increase the rendering performance but instead has wasted computational resources without a gain. The average costs C of a visibility test for object s using the bounding volume optimization may be formally described as

$$C(s) = C_B(s) + p_B(s)C_o(s), \tag{2.1}$$

where $C_B(s)$ describes the costs involved when testing the bounding volume surrounding s, $p_B(s)$ the probability that the bounding volume is visible, and $C_o(s)$ the costs of testing the object s for visibility.

When choosing the appropriate bounding volume one has to take into consideration that two

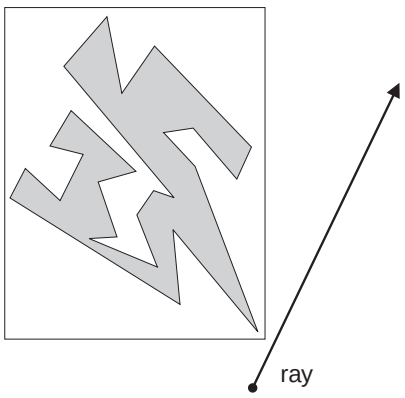


Figure 2.3 Complex object not hit by a tracing ray.

criteria influence the bounding volume's efficiency. On the one hand the void-area should be minimal, i.e. the bounding volume should fit the object as tight as possible to avoid unnecessary object tests (this will reduce the term $p_B(s)$). On the other hand testing the bounding volume for visibility should be as fast as possible (to minimize term $C_B(s)$). But these criteria contradict, since tight fitting bounding volumes tend to be hard to check, and vice versa. Thus, we have to use a compromise bounding volume.

2.1.2 Selected Bounding Volume Types

Frequently used bounding volume types include (see [MH99] for a complete dissusion of intersection algorithms for these bounding volumes):

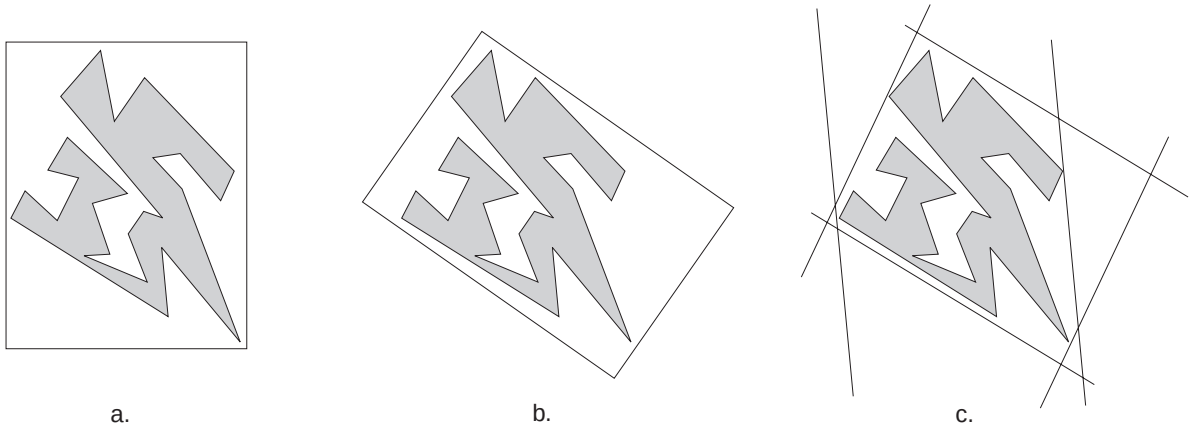


Figure 2.4 Bounding volume types: a.) axis-aligned box, b.) oriented box, c.) 8-DOP.

Axis-aligned Bounding Box (AABB): An *axis-aligned bounding box* is a box whose face normals coincide with the standard coordinate axes. (Figure 2.4a.)

Oriented Bounding Box (OBB): An *oriented bounding box* is a box where adjacent faces are orthogonal. (Figure 2.4b.)

Discrete Oriented Polytope (k-DOP): A *discrete oriented polytope* of degree k is defined by $k/2$ (k even) normalized normals n_i $(1 \leq i \leq k/2)$ and with each n_i two associated scalar values d_i^{min} and d_i^{max}, where $d_i^{min} \leq d_i^{max}$. Each triple $(n_i, d_i^{min}, d_i^{max})$ defines a slab S_i which is the volume between the two planes $\pi_i^{min} : n_i \cdot x + d_i^{min} = 0$ and $\pi_i^{max} : n_i \cdot x + d_i^{max} = 0$, and where the intersection of all slabs, $\bigcap_{1 \leq l \leq k/2} S_l$, is the actual k-DOP volume. (Figure 2.4c.)

Tribox: *Triboxes* [CR99] are a compromise between the efficient testing of AABBs and the tight fitting of general k-DOPs. A tribox is the intersection of bounding boxes formed in three different coordinate systems, each obtained by rotating the global coordiante system by 45 degree around one of the principal axes. In three dimensions a tribox has at most 18 polygonal faces. The authors of [CR99] present several optimization for this special case of a general k-DOP.

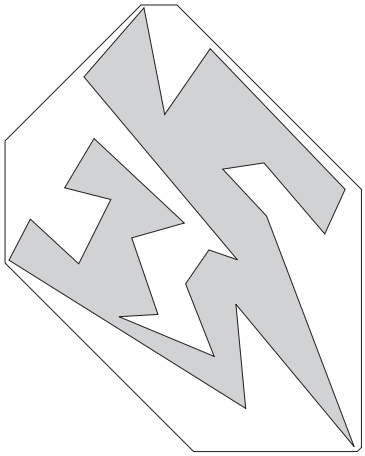


Figure 2.5 2-dimensional Tribox. A tribox is the intersection of bounding boxes formed in three different coordinate systems, each obtained by rotating the global coordiante system by 45 degree around one of the principal axes [CR99].

2.2 Hierarchical Bounding Volumes

It seems naturally to extend the basic idea of using bounding volumes to a hierarchical approach. Imagine how you would represent a city model in a scene data base: The city consists of many areas, these areas contain houses, houses can be subdivided into rooms, rooms contain furniture, etc. This defines an object hierarchy. If we assign a bounding volume to each hierarchy node we obtain a bounding volume hierarchy where an inner node of the related scene tree embrasses all bounding volumes of its children.

Concerning rendering algorithms, such a hierarchy can be used to speed-up visibility further. Starting at the root of the hierarchy nodes are tested for visibility as follows. If an inner node of the hierarchy is invisible descending the sub-tree is unnecessary since all of its child nodes must be invisible, too. If an inner node of the hierarchy is at least partially visible, we also recursively check its child nodes. These early rejection tests help to improve the rendering performance in many situations, but there is again the danger to introduce an unwanted overhead, i.e. if most of the scene objects are (partially) visible the hierarchical testing algorithm is no gain at all. Chapter 6 will present some ideas to eliminate this overhead.

2.3 Hierarchy Generation

Although a hand-modeled bounding volume hierarchy may improve the overall rendering performance compared to naive object-by-object visibility test, there is no guarantee that it may be useful for a certain rendering algorithm. Also, only few real-world models contain hierarchy information useful for the purpose of efficient rendering – if at all. This leads to the need of algorithms that create bounding volume hierarchies automatically.

2.3.1 Median Cut

Median Cut schemes such as [KK86a, Grö95] sort a set of n objects along one or more coordinate axes and build two new sub-sets of size $\lfloor n/2 \rfloor$ and $\lceil n/2 \rceil$ which comprise the first $\lfloor n/2 \rfloor$ and last $\lceil n/2 \rceil$ object in order. This process is repeated recursively which leads to the creation of a binary tree.

2.3.2 OBBTree

The *OBBTree* algorithm [GLM96] creates a binary tree which contains an OBB at each internal node and holds single objects in the leafs. The original algorithm handles only triangles as input objects. In a first step a close fitting OBB is searched. This OBB is subdivided along the longest axis of the OBB, which categorizes the triangles into individual sets depending on their relative location. In a recursive step for each of the the newly generated set of objects a new OBB is generated.

The challenging part of the algorithm is to find a well oriented OBB. As shown in the original paper [GLM96] this can be accomplished by computing an orientation of the triangles of the convex hull. Please refer to the original publications for details on this computation.

2.3.3 k-DOPTree

The *k-DOPTree* [HKM96, KHM+98] algorithm also creates a binary tree and demand for triangles as input. Again, a top-down approach is used to build the hierarchy. First, all the vertices of the set are projected onto each normal n_i of the k-DOP, the resulting extreme values (min,max) are stored in d_i^{min} and d_i^{max}. This gives the optimal k-DOP parameters for a given set. The original paper does not discuss the selection of normal vectors n_i.

After the k-DOP is found it will be subdivided into two subvolumes and the algorithm continues recursively. There are many potential ways to actually perform the subdivision. First the axis must be selected, along which the subdivision should take place. The most successful technique [KHM+98] computes the variance of the projections of the centroids of the triangles onto each axis and selects the axis that yields the one with the largest variance. Finally, the mean of the projected centroid coordinates determines the subdivision position. Triangles are assigned to the respective subvolumes based on their centroid location.

2.3.4 Bounding volume optimization

Goldsmith and Salmon [GS87] describe in the context of ray tracing a technique for automatic creation of bounding volume hierarchies. Their basic idea is to create a hierarchy that minimizes the approximated average intersection costs between a ray and a bounding volume hierarchy. The definition of a cost function (that is a hierarchical extension of Equation 2.1) allows to evaluate every given bounding hierarchy.

For hierarchy generation the authors propose an algorithm that incrementally adds objects to the initially empty hierarchy. New objects added to the hierarchy start at the root of the tree. Next, the subtree is determined that has the minimal bounding volume embrassing both the newly added object and all the objects within this subtree. This process is repeated until the

new object reaches a leaf of the tree. Finally, for every position along the path from the root to the leaf it is checked at which position adding the new object would minimize the overall hierarchy costs.

This scheme depends on the order in which objects are added to to hierarchy. The authors suggest to run the algorithm several times using random object order and select the resulting tree with minimal approximated costs.

2.3.5 Space Subdivision

Space-subdivision schemes like *octrees* [Gla84] or *bsp trees* [Jan86] are not object hierarchies in the strict sense. These techniques subdivide the space in which objects are lying in, rather than partitioning the set of objects in sub-sets. The space is subdivided into *voxels* or *half-spaces* depending on the underlying algorithms. Subdivided entities may contain multiple objects and objects may even be located within several subdivision entities.

CHAPTER 3

Hybrid Scene Structuring with Application to Ray Tracing

Abstract

The handling of highly complex 3D scenes is one of the major challenges in computer graphics. Several data structures were proposed in the past to address this problem. Many of these schemes are only suited for specific spatial distribution of objects in 3D space, making it difficult for a developer to select the appropriate data structure for the scene and/or application. Further, the selection of initialization parameters is typically a non-trivial task.

Using a ray casting environment this chapter presents an algorithm that automatically builds a hybrid data structure combining bounding volume hierarchies and uniform spatial subdivisions for a given scene. Our data structure is built by first creating a cost function based volume hierarchy, subsequently detecting regions of uniformly distributed objects using the scene hierarchy, and, as the last step, locally integrating uniform spatial subdivisions into the scene tree. We will show that the data structure can be built with low costs, both with regard to space and to run-time.

Finally, we present rendering times that demonstrate the usefulness of the new approach compared to standard techniques by comparing run-time efficiency. The memory requirements of the new data structure are, on average, linear in the number of scene objects. Further applications of the hybrid approach are also proposed.

3.1 Introduction

Ray tracing [Whi80] is a common technique for photorealistic visualization of three dimensional scenes. The high quality of rendered images is achieved by spending huge computational resources on the simulation of millions of light paths from the light sources to the eye of an observer. Many techniques have been developed to reduce the number of ray/object intersections by pre-structuring the scene [AK89].

A common algorithm to reduce the number of required intersection tests is the intersection of rays with a hierarchy of bounding volumes, rather than with the whole set of objects [KK86a, GS87] (Figure 3.1). A performance gain is achieved because one can conclude that if the bounding volume of a subscene is not hit by a ray, no object within the subscene can be hit. Creation of good bounding volume hierarchies is a difficult problem, though, since the number

of possible hierarchies grows exponentially with the number of objects.

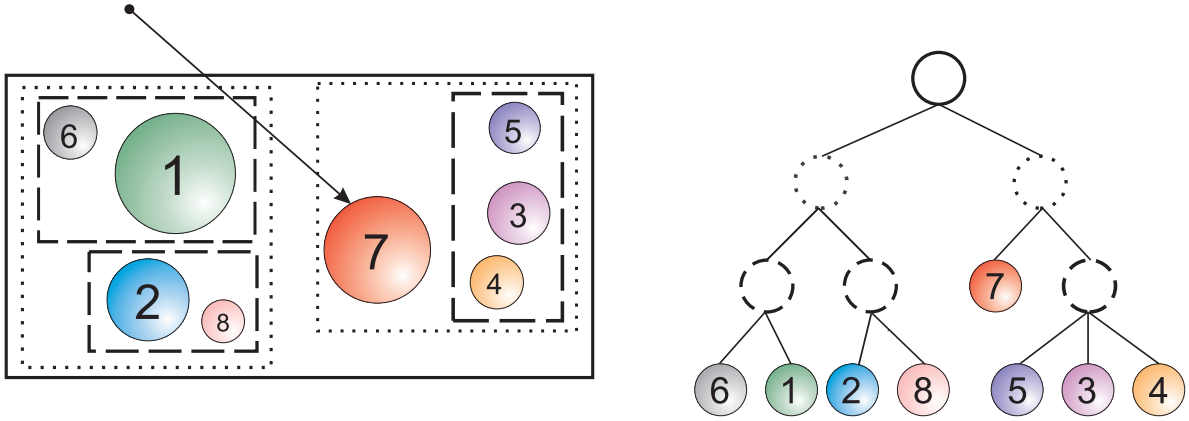


Figure 3.1 Hierarchy of bounding volumes.

Space subdivision techniques sudivide the space enclosing a scene into non-overlapping regions, called *voxels*. For each voxel a list of objects is created describing the objects that intersect the voxel. When a ray passes through space, only those objects that lie in voxels intersected by the ray must be tested to detect ray/object intersections. Depending on the different space subdivision strategies one distinguishes between regular and adaptive techniques [FTI86, Gla84, Jan86, CDP95] (Figure 3.2).

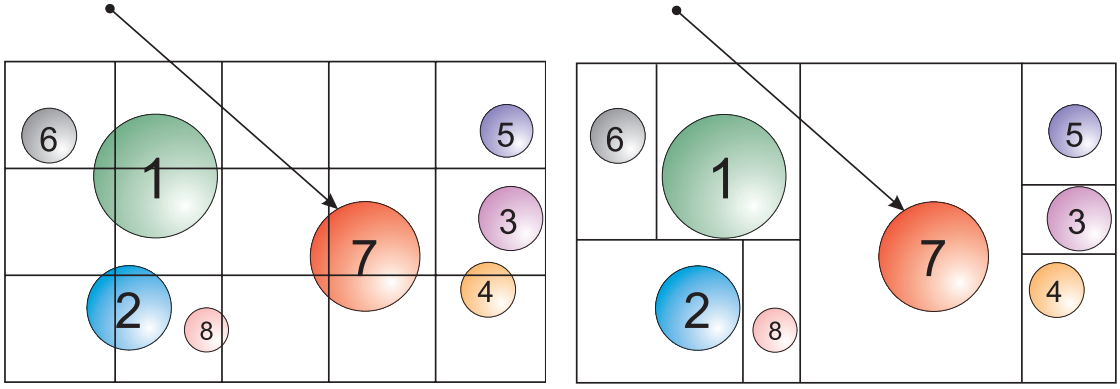


Figure 3.2 Uniform and adaptive spatial subdivision.

Because ray tracing acceleration schemes differ in their relative efficiency from scene to scene, no single technique can be optimal under all circumstances [Arv90]. Bounding volume hierarchies are well suited for inhomogeneous scenes, uniform space subdivisions are well suited for scenes with homogeneous object distribution in space, while adaptive space subdivision are well applicable to many types of scenes without giving optimal efficiency [Mül97]. These results led us to the development of a hybrid scheme combining the advantages of bounding volume hierachies and uniform subdivision. Results proof that this new technique outperforms each of the single techniques.

An example of a resulting data structure is shown in Figure 3.3: A scene is partioned into two (possibly distant) subscenes using a bounding volume hierarchy. One subscene contains many subscenes uniformly distributed in space, thus we are building a uniform spatial subdivision to represent the subscene. Note, that highly inhomogeneous detail contained in one of the resulting voxels, could again be modeled using a hierarchy of bounding volumes.

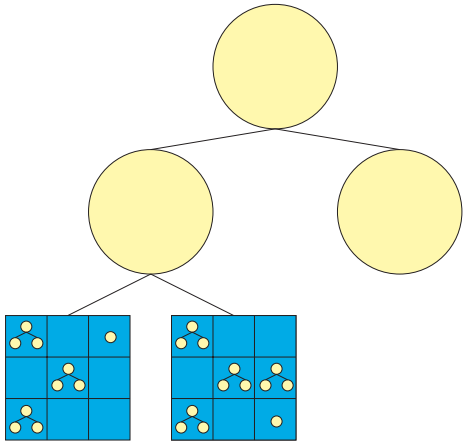


Figure 3.3 Combining bounding volumes hierarchies and space subdivisions.

3.2 Hybrid Scene Structuring

In this section we will present our new scene structuring algorithm that is best described by three subsequently performed stages:

1. In the first stage, we create a bounding volume hierarchy for a given scene of objects. The basic idea is close to the median cut scheme described in [KK86a] which recursively computes partitions of the objects in two equally sized subsets based on their spatial location relative to a coordinate axis. We extend that scheme by introducing a cost function similar to [GS87] to gain better scene partitions.

2. Stage 2 of our algorithm detects scene regions in which objects have similar size, have close-by neighbors, and are uniformly distributed in space. Such regions (identified by testing the bounding volumes of subscenes within the scene hierarchy) are marked for later usage. This detection phase makes heavy use of hierarchy information created in stage 1 of our algorithm. The different levels of the tree (corresponding to the scene hierarchy) hold precious information on object distribution at multiple levels of detail.

3. Finally, we create uniform spatial subdivisions for those nodes of the scene tree that were marked in the previous stage of the algorithm. Again, the hierarchical scene representation can be used to speed-up the space partioning significantly.

3.2.1 Hierarchical Bounding Volume Optimization

Our scene hierarchy tree is built top-down by recursively subdividing the set of scene objects into two disjoint sub-sets. Each node of the corresponding binary tree represents a specific subscene of the whole scene and stores an axes-aligned bounding box of this subscene.

At each subdivision step we sort the objects along all coordinate axes. The center of an object's bounding box serves as the sorting key in this process. Next, we are evaluating several potential subdivision positions along each coordinate axis by splitting the sorted list of objects into a *left* and *right* part. This is similar to the median cut scheme presented in [KK86a], but we don't have a predefined split position. Instead, we minimize a cost function describing the approximated costs for computing the ray/scene-intersection for a specific subdivision position by testing *all* possible split positions in the sorted list of objects. Subdivision stops if the intersection cost associated to a subscene falls below a predefined threshold.

Our cost function is based on the observation that – assuming an uniform distribution of rays – the conditional probability that a ray will intersect a convex bounding volume B given that it intersects a surrounding convex bounding volume A is $\frac{S(B)}{S(A)}$, with S representing a bounding volume's surface area [GS87]. The costs of intersecting a ray with a subscene are approximated by summing the invidial intersection costs predetermined for each object type. This leads directly to the definition of our cost function C describing the approximated intersection cost of a ray with a given scene hierarchy H with direct children L_j and R_j. L_j and R_j result from a subdivision of the sorted object list covered by H ($|H| = n$)[1] into two sub-lists by performing a split along a given coordinate axis ($|L_j| = j$, $|R_j| = n - j$; $j \in \{1, 2, \ldots, n-1\}$):

$$\begin{aligned} C_H(j, axis) &= \frac{S_B(L_j)}{S_B(H)} \cdot \sum_{i \in L_j} C_{obj}(o_i) \\ &\quad + \frac{S_B(R_j)}{S_B(H)} \cdot \sum_{i \in R_j} C_{obj}(o_i) \end{aligned} \tag{3.1}$$

with

- $C_{obj}(o)$ being the average ray/object intersection costs intersecting a random ray with elementary object o
- $S_B(X)$ being the surface area of the bounding box associated to (sub)scene X
- $axis \in \{X, Y, Z\}$

When performing a subdivision step at an inner node of the hierarchy, the cost function must be evaluated for all potential subdivision positions and coordinate axes. For a single coordinate axis and n objects, this needs $2 \cdot (n-1)$ bounding volume computations (which can be done in time $O(n)$ by incrementally unifying bounding volumes assigned to the elementary objects). The sorting of all objects should be done in a pre-sorting process for each coordinate axis, since the relative positions of objects will not change along an axis. Thus, the run-time costs for a single subdivision step is linear in the number of objects to be subdivided. Assuming random split positions, this leads to a overall run-time complexity of $O(n \log n)$ in the average case.

[1] $|H|$ is the number of objects stored in the leafs of hierarchy H.

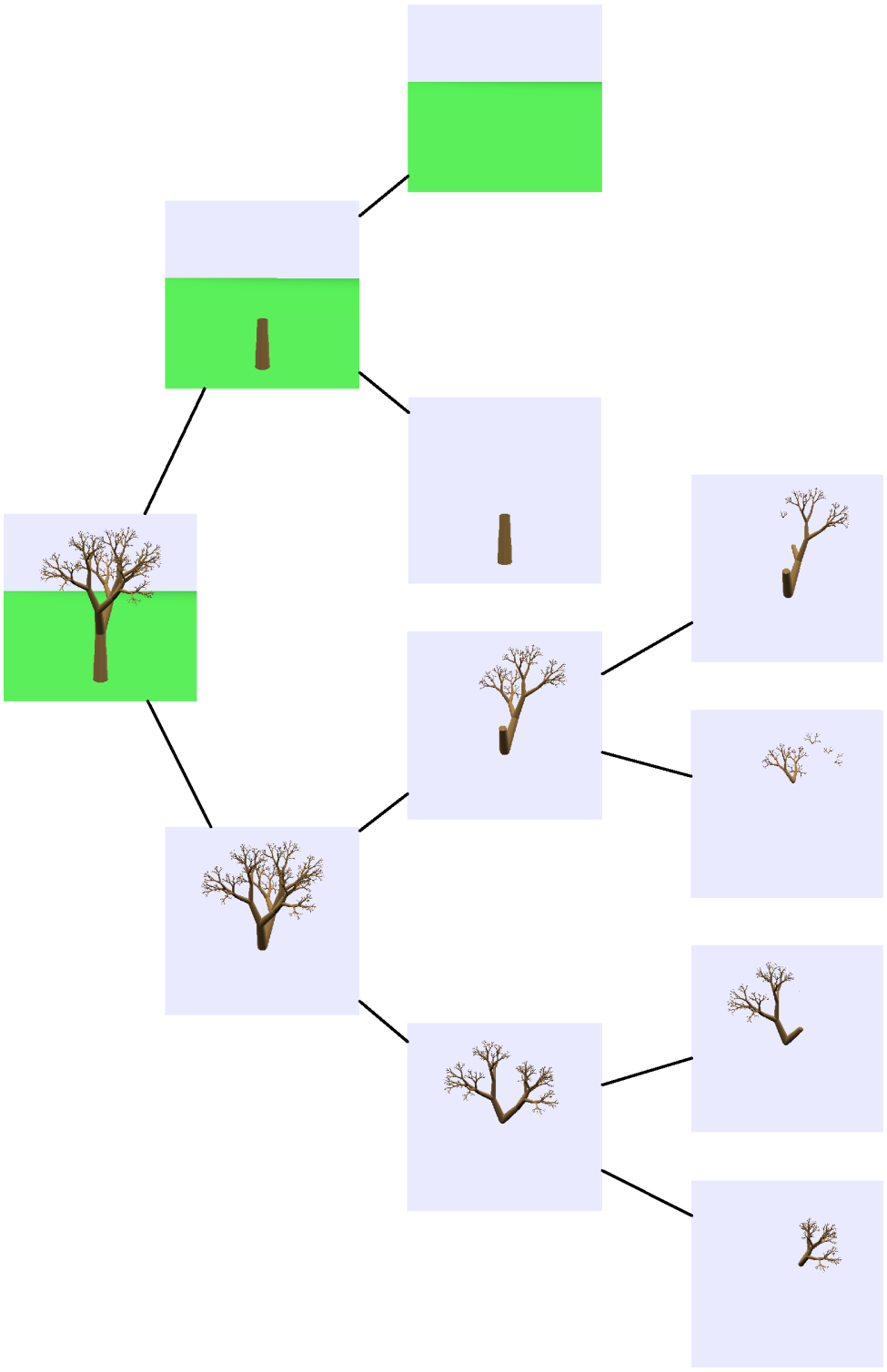


Figure 3.4 Scene structuring using a cost function based bounding volume optimization.

```
void Node::classify()
{
    m_uniform_base = false;
        // no space subdivision by default
    m_avgObjSize = boundingVolumeDiameter();
    if (leaf()) {
        m_uniform_base = true;
        m_voxels = 1;
        return;
    }
    Node* son1 = leftSon(); son1→classify();
    Node* son2 = rightSon(); son2→classify();

    if (compact(son1,son2) == true) {
        m_uniform_base = true;
        if ( objSizeSimilar(son1,son2)
            && son1→m_uniform_base == true
            && son2→m_uniform_base == true) {
                unsigned vox1 = son1→m_voxels;
                unsigned vox2 = son2→m_voxels;
                m_voxels = vox1 + vox2;
                double w1 = vox1 * son1→m_avgObjSize;
                double w2 = vox2 * son2→m_avgObjSize;
                m_avgObjSize = (w1 + w2) / m_voxels;
                son1→m_uniform_base = false;
                son2→m_uniform_base = false;
        } else {
            m_voxels = 2;
        }
    }
    return;
}
```

Figure 3.5 Detecting scene nodes/regions suitable for uniform space subdivision and marking subscene corresponding nodes by setting *m_uniform_base true*.

Figure 3.4 shows the top levels of a scene hierarchy built with the presented algorithm. Each node of the tree visualizes the subscene that is bounded by the corresponding node of the generated hierarchy. For the sample scene in Figure 3.4, most ray/scene intersections happen with the floor area of the visualized landscape. This is taken into consideration by automatically placing the floor area close to the root of the hierarchy, thus, testing it very early when doing a ray/scene intersection.

3.2.2 Classifying Scene Nodes

In the second stage of our algorithm we want to detect possibly large scene regions that are best structured by a uniform spatial subdivision of the scene, i.e. regions with objects of similar size, uniform distribution of objects within the region, and small empty spaces between neighbor objects. Regions that do not have these characteristics will be left unchanged, i.e. structured using the bounding volume hierarchy created in the previous stage of the algorithm. Our algorithm

```
bool Node::compact(Node* son1, Node* son2)
{
    Box a = boundingBox(),
        a1 = son1→boundingBox(),
        a2 = son2→boundingBox();
    double s = a.surfaceArea(),
        s1 = a1.sufaceArea(),
        s2 = a2.surfaceArea();
    double v = a.volume(),
        v1 = a1.volume(),
        v2 = a2.volume();

    const double c_s = .250;    // suitable
    const double c_v = .125;    // compactness parameters

    if ((s1+s2) < c_s * s) { return false; }
    if ((v1+v2) < c_v * v) { return false; }
    return true;
}
```

Figure 3.6 Testing spatial neighborhood between subscenes.

```
bool objSizeSimilar(Node* son1, Node* son2)
{
    const double c_d = .3;    // similarity
    double size1 = son1→m_avgObjSize;
    double size2 = son2→m_avgObjSize;

    if (size1 < size2) {
        return (size1 > c_d * size2);
    } else {
        return (size1 < c_d * size2)
    }
}
```

Figure 3.7 Comparing average object sizes between subscenes.

marks scene nodes dependent on the fact if they are suitable base nodes for a uniform subdivision for the corresponding subscene (Figure 3.5). Additionally, the algorithm also computes an appropriate number of subdivision voxels for all marked nodes/subscenes.

The classification method is quite simple: First, resursively classify all subscenes. (By definition all leafs will be marked.) Then, check if the two bounding volumes of an inner node's direct sons are *compact*[2] (Figure 3.6). Finally, mark the current node based on the result of testing the bounding box for compactness. If the boxes are not compact, the subscenes have a large distance to each other and a uniform subdivision is counter-productive: the node will not

[2] We define two boxes B_1 and B_2 to be *compact*, iff a) the summed surface area of B_1 and B_2 is not larger than the surface area (times c_s) of the smallest box B bounding B_1 and B_2 and b) the summed volume of B_1 and B_2 is not larger than the volume (times c_v) of B ($c_s, c_v > 0$). If two boxes are not compact they have a large spatial distance.

be marked. Otherwise – the boxes are compact – we have to check the sons, provided they are also marked and contain objects of similar size (measured with the bounding volumes' diameters). If this is the case, we have detected two uniform regions that, again, may be combined to a larger uniform region. Thus, we delete the marks associated with the sons, indicating that we have found a new base for uniform subdivision. The number of voxels associated to a uniformly subdividable subscene is set to the sum of the number of voxels associated to each son. If the sons are compact, but at least one of them is not marked or the objects contained in the corresponding subscene have significantly different sizes (by a factor of c_d, Figure 3.7), we treat the current node as an elementary object in any subscene it is contained. This enables us to build a (typically small) uniform subdivision over a set of subscenes that lie close together, but have highly non-uniform inner structure.

Note, that our algorithm will not destroy the underlying scene hierarchy within a region to subdivide. We will use the hierarchy in the next stage of scene structuring. Also note, that the number of voxels associated to each marked node equals the largest possible number of scene nodes (which may be leafs or inner nodes) that are uniformly distributed below a marked scene node.

3.2.3 Locally Uniform Space Subdivision

The final stage of our scene structuring algorithm will show how to build uniform space subdivisions for those subscenes that were marked in the previous step. Assume we have a marked node with m being the number of space subdivision voxels assigned to this node. We create a uniform space subdivision with m voxels by uniformly subdividing the axes-aligned box enclosing the subscene represented by the marked scene node. Similar to [Spa90] and [Grö95], we create the space subdivision by recursively subdividing the bounding box. At each step, we subdivide all voxels along the dominant coordinate axis into two equally sized voxels and compute those objects that have a non-empty intersection with those voxels. The recursion stops if the pre-determined number of voxels is reached.

At each voxel subdivision step, we have to compute those objects that intersect a given voxel. Using scene hierarchy information gained in Section 3.2.1, we can speed-up the voxel membership tests significantly:

- If the bounding volume of a scene node lies completely inside the current voxel all subscenes must also lie within the voxel.
- If the bounding volume of a scene node lies completely outside the current voxel all subscenes must also lie outside the voxel.

In both cases, tests with subscenes can be omitted. Only in the case that a bounding volume of a voxel is partially inside and partially outside the current voxel, each direct sub-node must be tested for membership recursively. When storing the objects lying completely inside a voxel we store scene nodes whenever possible instead of lists of elementary objects to preserve hierarchy information in the recursion.

Because of the regular structure of the subdivided space, we can expect that, on average, a single voxel split operation can be done in constant time, independent of the number of scene

objects. At each subdivision step, we test voxel membership for the top nodes of those subscenes being left after the previous recursion level. Their bounding volume size is very similar to the size of the current voxel because of uniform distribution of the objects (and the subscene bounding volumes containing the objects). Thus, for a single uniform voxelization with k voxels we need only time $O(k)$, if a single voxel subdivision can be done in time $O(1)$ on average. Since marked subscenes are treated as elementary objects if they lie within a larger marked subscene, all uniform regions of a scene can then be subdivided in time $O(n)$ using the scene hierarchy.

This is a remarkable fact, because it implies, that the whole scene voxelization can be done in a very short time, provided the scene hierarchy from Section 3.2.1 is available. We could compute the scene hierarchy *once*, save the result to an external storage medium, and perform a fast voxelization based on the precomputed scene hierarchy, whenever using the scene.

As already mentioned before, we perform a uniform space subdivision for each marked node. The number of subdivision voxels equals the number associated to each of these nodes. Since we are subdividing space within which objects or subscenes are distributed uniformly, the space to store a subdivided region will be approximately linear in the number of voxels. Note, that a marked scene node within a larger marked subscene only counts as one elementary object. For this reason, the overall space time complexity, on average, is $O(n)$ (n = number of elementary objects).

3.3 Results

We have compared absolute rendering times to visualize several test scenes (Figures 3.8 and 3.9) with different ray-acceleration techniques in a ray-tracing framework (Table 3.1). Measured times are split into *setup* (creation of data structures in a preprocessing step) and *render* (computing ray/scene intersections and shading). Table 3.1 also includes the accumulated number of ray/object intersections and intersection tests. The test scenes were taken from the the SPD package [Hai87] and cover different kinds of object distributions in space making them a suitable testbed for our performance measurements. All measurements were done on a PC having a 300 Mhz Pentium II CPU and 64 MB of main memory.

3.3.1 Description of Compared Acceleration Techniques

BVolC: Cost function based hierarchical bounding volume creation described in Section 3.2.1.

BVolM: Similar to **BVolC** but only evaluating cost function C at subdivision position $j = \lfloor n/2 \rfloor$ (median cut).

RegSub: Uniform space subdivision of the whole space covered by the scene's axes-aligned bounding box (10,000 voxels). Voxels are distributed in space such that the number of voxels in x, y, and z direction is roughly proportial to the scenes bounding volume extents in these dimensions.

RegSub*: Same as **RegSub**, but using 40,000 instead of 10,000 voxels.

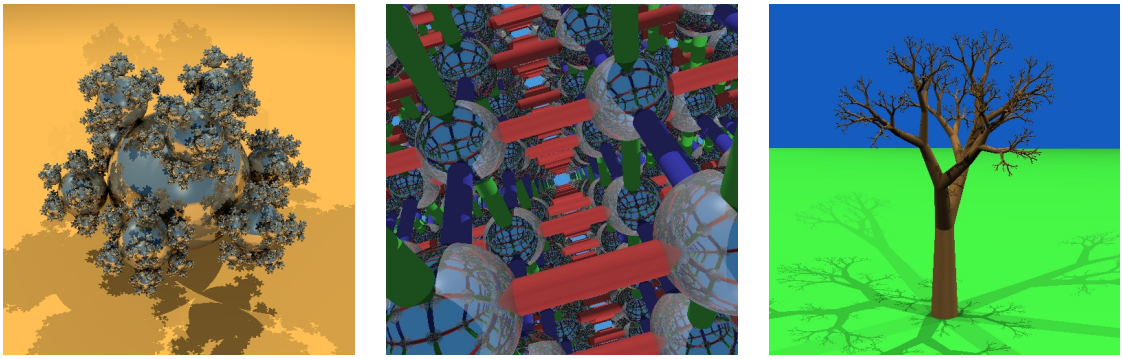


Figure 3.8 Test scenes *balls*, *lattice*, and *tree*.

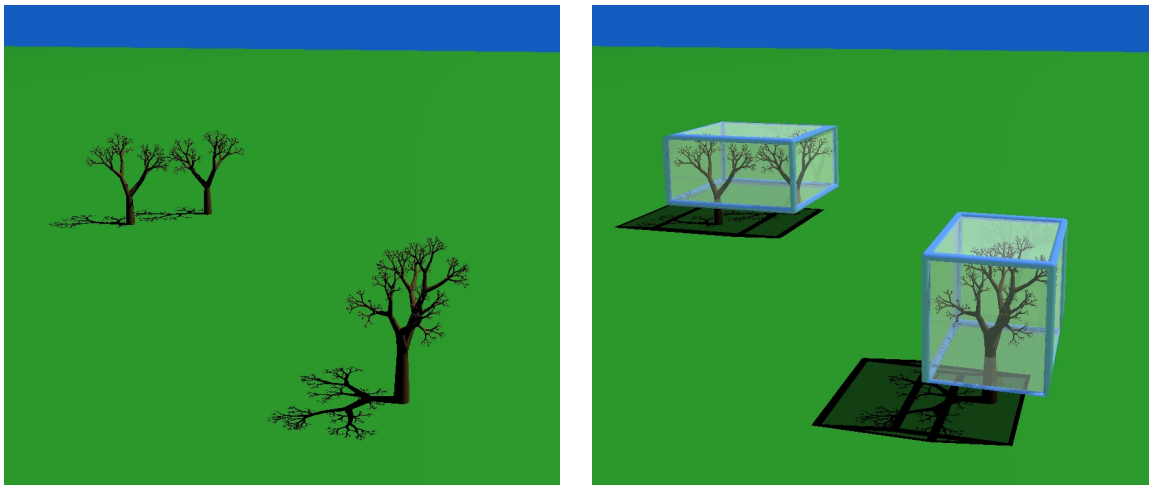


Figure 3.9 Test scene *trees*. The right picture shows the scene regions that are uniform subdivided by method **Hybrid**.

HVoxel: Same as **RegSub**, but in a preprocessing step a **BVolC**-hierarchy is created. When doing space subdivision, a voxel that covers a subscene completely stores a subscene node instead of a list of elementary objects. This may drastically reduce the run- and space-time cost for the case that many elementary objects fall into a single voxel by using the hierarchy information available.

BSP: Binary space partitioning scheme which recursively subdivides the scene's bounding box into 2 equally sized sub-boxes [SS92]. The maximum subdivision depth was chosen to be 18, recursion is also stopped when a region contains less than 4 elementary objects.

KDTree: Similar to **BSP** but the subdivision plane is selected to minimize a cost function very similar to function C from Section 3.2.1. The costs of a **BSP**-node are defined as the summed object intersection costs of the objects contained in the 2 sub-boxes weighted by the surface area of the sub-box they are contained in [Mül97]. In a preprocessing step the objects are sorted along all coordinate axes for efficient cost function minimization.

Octree: Space-subdivision scheme that subdivides the scene's bounding box recursively into 8 equally sized sub-boxes. The maximum subdivision depth was chosen to be 6, recursion is also stopped when a region contains less than 4 elementary objects.

Octree*: Same as **Octree**, but maximum subdivision depth was changed to 8.

HUG: Hierarchy of Uniform Grids [CDP95]: Objects are filtered by size yielding 3 equally sized sets/levels of objects. Then, objects in each set are clustered to identify spatially distant sub-sets which are subdivided using an uniform grid. Finally, a hierarchy of these grids is built by inserting grids containing smaller elementary objects into intersected higher level grid cells containing larger elementary objects.

HUG*: Same as **HUG**, but voxels of each grid are distributed in space such that the number of voxels in x, y, and z direction is roughly proportional to the scenes' bounding volume extends in these dimensions.

Hybrid: Our hybrid acceleration technique described in Section 3.2.

3.3.2 Comparision of Run-Time Efficiency

Comparing **BVolC** and **BVolM** we see that the cost function based hierarchy construction scheme is superior to a simple median cut scheme concerning rendering efficiency. Comparing setup times, **BVolC** is slightly slower that **BVolM** mostly because the cost function must be computed for many potential split positions. The median cut scheme has a worst and best case complexity of $O(n \log n)$ while **BVolC** has only an average case complexity of $O(n \log n)$, assuming random subdivision positions.

The chain of argumentation is similar when comparing the adaptive space subdivision schemes **BSP** and **KDTree**. Again, it seems to be worth to spend more time in the preprocessing phase to find good subdivision positions. This overhead is mostly compensated in the render phase.

Results for the **HUG**-scheme are rather disappointing for the test scenes *balls*, *tree*, and *trees*. The main problem seems to be that the filter technique used is simply to naive (3 levels, same number of objects each) to guarantee $O(1)$ objects per grid cell, which is a must for efficient space subdivision techniques. On the other hand, the overall technique sounds very promising. Further research has to concentrate on finding more appropriate filter functions.

When comparing run-time results for **BVolC** and **RegSub** one can easily see that **BVolC** is better suited for inhomogeneous object distributions (*balls*, *tree*) in space than **RegSub**. On the other hand **RegSub** is an appropriate scheme for regulary structured scenes like *lattice*. The **RegSub** results can be improved by applying the **HVoxel**-optimization, though, which guarantees that for test scenes *balls*, *tree*, and *trees* a hierarchical scene structure for densely populated voxels is generated.

Finally, comparing our new technique **Hybrid** with the other schemes presented, one can easily see the benefit of the new approach. For all test scenes we can further reduce the rendering times compared with the other schemes – sometimes even significantly (*balls*). Even, if the rendering times were of similar magnitude compared with the fastest of the other techniques, our algorithm has the advantage that no user interaction is needed to select an appropriate scheme together with its initialization parameters. A drawback of our scheme is the fact that it has

balls (7383 objects)			
method	intersections	setup (sec)	render (sec)
BVolC	42,526,503	1.58	52.51
BVolM	80,248,331	1.36	97.79
RegSub	374,426,211	0.60	264.23
RegSub*	142,908,934	1.06	111.14
HVoxel	78,032,975	1.68	80.93
BSP	89,572,197	0.60	118.25
KDTree	21,073,145	3.54	39.18
Octree	168,124,433	0.50	189.67
Octree*	16,456,617	0.76	82.76
HUG	352,291,425	2.13	281.65
HUG*	290,883,945	2.29	154.62
Hybrid	13,293,618	2.59	26.65

lattice (8281 objects)			
method	intersections	setup (sec)	render (sec)
BVolC	136,627,223	2.27	207.67
BVolM	157,338,109	2.08	296.60
RegSub	18,682,257	2.31	47.34
RegSub*	21,015,736	3.67	41.85
HVoxel	12,955,345	5.56	38.34
BSP	14,506,553	3.62	64.08
KDTree	14,621,822	5.73	63.52
Octree	10,680,034	4.68	100.06
Octree*	10,621,667	5.89	101.20
HUG	17,831,280	3.96	46.62
HUG*	17,831,280	4.12	46.54
Hybrid	12,159,322	5.63	39.45

tree (8192 objects)			
method	intersections	setup (sec)	render (sec)
BVolC	15,692,221	2.01	22.57
BVolM	44,714,514	1.76	47.31
RegSub	1,144,488,683	1.73	2258.38
RegSub*	432,183,008	2.25	805.23
HVoxel	76,184,307	5.56	38.34
BSP	343,908,664	1.71	834.06
KDTree	8,175,353	5.68	34.26
Octree	723,241,229	1.95	1616.75
Octree*	56,728,797	2.37	172.43
HUG	1,197,585,510	3.87	2694.29
HUG*	681,488,036	3.98	1387.39
Hybrid	9,923,958	5.07	21.10

trees (24572 objects)			
method	intersections	setup (sec)	render (sec)
BVolC	3,413,744	7.32	6.98
BVolM	13,977,722	6.21	18.70
RegSub	538,128,376	6.80	1119.92
RegSub*	205,420,337	7.70	435.35
HVoxel	11,048,194	9.31	22.65
BSP	379,922,862	5.37	788.56
KDTree	1,540,374	11.14	7.35
Octree	767,520,736	7.39	1666.09
Octree*	50,222,304	8.84	108.48
HUG	153,262,282	5.34	211.45
HUG*	153,262,172	5.23	211.68
Hybrid	2,888,736	9.01	7.00

Table 3.1 Run-time statistics for ray tracing several different scenes using different acceleration techniques.

slightly larger initialization costs on average, but these amortize if larger number of rays have to be traced, either because a larger image resolution is selected or a more complex ray integration technique is used.

3.4 A Framework for Scene Structuring

The algorithms described in the previous sections were implemented using the MRT rendering package (Minimal Rendering Tool) [Fel96], an extensible library of rendering algorithms and data structures. Through an object-oriented interface its functionality can be accessed at several levels of abstraction to suit the needs of different applications.

MRT's base class for all geometric objects is class t_Object. Important virtual messages of class t_Object include boundingVolume() which returns the bounding volume of an object, intersect() computes the intersection between a ray and the object, rayIntersectCosts() compute the average intersection cost between a ray and the object, and checkBoxIntersect() is provided for the initialization of spatial data structures by testing if the object has an non-empty intersec-

tion with an axes-aligned bounding box.

A remarkable fact of the MRT design is that the whole scene as well as all subscenes within a hierarchy are handled consistently as elementary geometric objects, i.e. class t_Scene is derived from class t_Object and thus also provides the above messages. To compute the intersection of a ray with a scene, the scene first tests for an intersection with its bounding volume and then sends message intersect() to all objects it contains which can either be elementary objects or scenes. Introducing new acceleration schemes is very simple by directly deriving from class t_Scene and overloading methods of t_Object like intersect(). Because all scene structuring schemes share the interface with classes t_Scene and t_Object, it is possible to combine different acceleration schemes within a single hierarchy of scene representations.

3.5 Conclusion and Further Work

We have presented a new data structure for ray tracing complex scenes. We have shown that ray tracing computation times can be significantly reduced using the new scheme compared to many other acceleration schemes. As a key feature, the new data structure does not need any user supplied setup parameters to initialize properly, thereby outperforming all competing algorithms. The constants c_s, c_v, and c_d used in our definitions of compactness and similarity of bounding volumes have been determined empirically by evaluating many test scenes. We have found these constants to be independent of the actual scene geometry. Future work will check this out in more detail.

Recently, independently of our work, Klimansezewski and Sederberg have presented a ray accelleration method [KS97] that uses a bounding volume hierarchy to control the construction of an adaptive grid structure. They use a different bounding volume creation technique [GS87] and a different heuristic from ours to decide which sub-grids have to be merged to build a larger grid. Though, a more detailed comparision of the two methods has yet to be done.

Beyond the scope of this chapter we are also testing the usefulness of the new scheme for applications other than ray tracing. We have prelimenary results indicating that the automatic creation of object hierarchies presented in this chapter produces bounding volume hierarchies that are well suited for many applications ranging from hierarchical view frustrum culling to occlusion testing and collision detection in complex scenes.

We will also test the hybrid data structure as a scene structuring tool for a hierarchical radiosity algorithm using object clusters. Our bounding hierarchies will define a natural scene hierarchy while the uniformly subdivided sub-spaces represent object clusters.

CHAPTER 4

Automatic Creation of Object Hierarchies for Radiosity Clustering

Abstract

Using object clusters for hierarchical radiosity greatly improves the efficiency and thus usability of radiosity computations. By eliminating the quadratic starting phase very large scenes containing about 100k polygons can be handled efficiently. Although the main algorithm extends rather easily to using object clusters, the creation of 'good' object hierarchies is a difficult task both in terms of construction time and in the way how surfaces or objects are grouped to clusters. The quality of an object hierarchy for clustering depends on its ability to accurately simulate the hierarchy of the energy flow in a given scene. Additionally it should support visibility computations by providing efficient ray acceleration techniques.

In this chapter we will present a new approach of building hierarchies of object clusters. Our hybrid structuring algorithm provides accuracy and speed by combining a highly optimized bounding volume hierarchy together with uniform spatial subdivisions for nodes with regular object densities. The algorithm works without user intervention and is well suited for a wide variety of scenes. First results of using these hierarchies in a radiosity clustering environment are very promising and will be presented here.

The combination of very deep hierarchies (we use a binary tree) together with an efficient ray acceleration structure shifts the computational effort away from form factor and visibility calculation towards accurately propagating the energy through the hierarchy. We will show how an efficient single pass gathering can be used to minimize traversal costs.

4.1 Introduction

Clustering for hierarchical radiosity [Sil95, SAG94] works by grouping patches or objects to clusters and performing approximative energy exchange between these clusters. Starting with a single self-link the algorithm checks if a given link accurately represents the energy exchange between two object clusters or if it has to be refined. Link refinement means deleting the current link and establishing new links to or from the next level of the object hierarchy, thus defining a recursive refinement process. Once the surface level of the hierarchy is reached a standard hierarchical radiosity algorithm proceeds.

After introducing clustering algorithms for radiosity computations enhancements to the ra-

diosity algorithm like incorporating glossy reflection [CLSS95] and the efficient use of error-bounds [GH96, SSS98] have been published. Recent research however, focuses on the efficient and accurate creation of object hierarchies that can be used as a starting point for the radiosity calculation.

Hasenfratz et al. [HDSD99] give a thorough analysis of different clustering algorithms used for radiosity computations. Their investigation was driven by two issues. First, the quality of the simulation, i.e. the accurate flow of energy, and second, the overall application performance which is typically dominated by the ray casting costs due to visibility checks. Attention must also be paid to the cluster creation time, which can be significant if an elaborate optimization step is involved. Comparing different clustering strategies by applying them to 'real-world' scenes, i.e. scenes modeled for other purposes than radiosity calculations, they made the following observations: Clustering schemes based on hierarchical bounding volumes typically behave the most predictable but do not always result in best performance. The combination of a k-d tree with a local bounding volume optimizations in densely populated inner nodes however, gives good ray acceleration results, thus probably improving overall performance (but not necessarily image quality). Finally, bottom-up construction methods generally produce better object hierarchies although the costs for the creation step can easily become quadratic.

Led by these observations we will present a new clustering algorithm that can be characterized by the following properties:

- hierarchical bounding volumes in a binary tree
- fast creation time ($O(n \log n)$)
- reliable detection of original scene objects
- efficient ray-acceleration

The structure of our chapter is as follows. In the next section the clustering algorithm will be explained. Section 3 describes our radiosity implementation and finally we will present and discuss our results.

4.2 Finding Object Clusters

4.2.1 Overview

Our new clustering algorithm creates a hierarchy of bounding volumes which define the individual clusters for the radiosity process.

The algorithm examines the existing objects and their mutual spatial relationship. This yields very high quality hierarchies with structures which are often called *intuitive* or *natural*. The cluster hierarchies adapt well to the spatial distribution of objects:

- Spatially separated geometry is identified and placed into separate clusters.
- Overlaps between clusters are reduced and tight bounding volumes are built by finding bounding volumes with minimal surface areas.

The basic idea behind the construction scheme is close to the median cut scheme described in [Grö95, KK86a] which recursively computes partitions of the objects in two equally sized subsets based on their spatial location relative to a coordinate axis. We extend this scheme by introducing a cost function similar to [GS87] to gain significantly better scene partitions. Also, we will show that the data structure can be built very efficiently.

4.2.2 Construction

Our hierarchical bounding volume optimization method subdivides recursively the set of scene objects into two disjoint subdivision entities. Each node of the corresponding binary tree represents a specific subscene of the whole scene. At each subdivision level, the individual objects are assigned to exactly one subdivision entity of that level. No objects are split, hence no new objects (e.g. polygons, triangles, etc.) are created by this method.

Starting from the root node, we sort the objects along all major coordinate axes, where the center of an object's bounding box serves as sorting key. Based on these sorted lists, we evaluate potential partitioning positions along each axis for each entry in the respective list by splitting the sorted list of object into a *left* and *right* part. In contrast to the median cut scheme applied in [Grö95], we don't have a predefined partitioning position. Instead, we minimize a cost function describing the approximated costs computing the ray/scene-intersection for a specific subdivision position. By minimizing the cost function over *all* partitioning positions, we obtain an optimal subdivision position which generates two new subdivision entities; one containing the *left* objects, one containing the *right* objects (Figure 4.1). The subdivision process terminates for subscenes which contain only a single object.

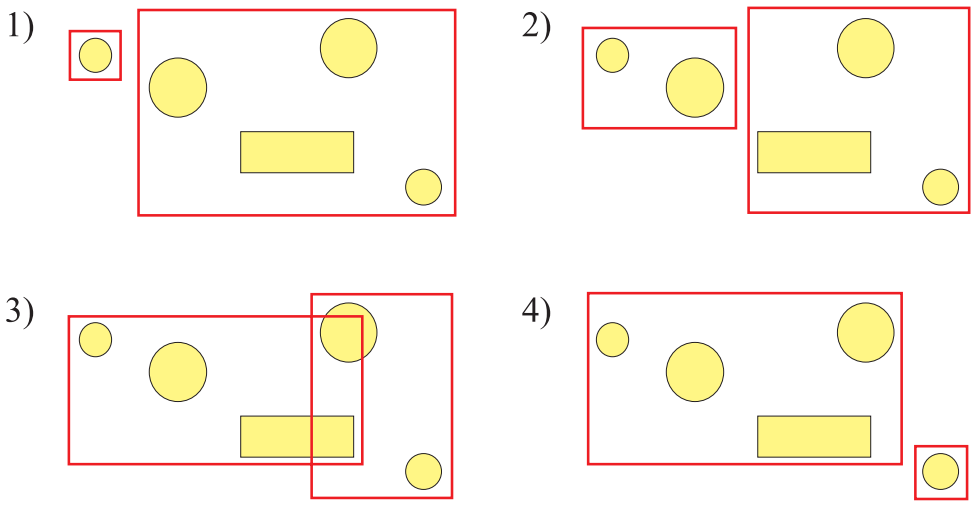


Figure 4.1 Possible object partitions along a single coordinate axis for a simple example scene containing 5 objects. The partition with minimal costs according to function C will be used in the subsequent recursion step.

The cost function used has already been successfully applied in a ray-tracing environment and has shown its superiority compared to other bounding volume schemes. The costs of a

subdivision entity H, with children H_{left} and H_{right}, is given by:

$$C_H(axis) = \frac{S(H_{left})}{S(H)} \cdot |H_{left}| + \frac{S(H_{right})}{S(H)} \cdot |H_{right}|$$

with

- $|H|$ the number of objects within hierarchy H
- $S(H)$ being the surface area of the bounding box associated to scene H
- $axis \in \{X, Y, Z\}$

Applying hierarchies based on this cost function to radiosity clustering is primarily a consequence of two observations:

- An efficient ray acceleration scheme is needed for fast visibility computations based on ray casting.
- Results in [MF99] show that this algorithm generates hierarchies that adapt well to the distribution of objects in the scene. E.g. polygons of individual objects are detected and clustered together, and overlaps of bounding volumes are minimized (although overlaps are still possible).

Construction Costs

Many previously proposed bounding volume schemes fail for large scenes because of the high computational complexity involved in the respective construction method. Their complexities easily become quadratic by checking mutual spatial relationship making them unusable for large geometries composed of many objects. Our method will need only time $O(n \log n)$ in the average case while guaranteeing hierarchies of very high quality comparable to bottom-up methods.

When performing a subdivision step at an inner node of the hierarchy, the cost function has to be evaluated for all potential subdivision positions and coordinate axes. This can be done in time linear in the number of objects by incrementally unifying bounding volumes assigned to the elementary objects. The sorting should be done in a pre-process for each coordinate axis, since the mutual relative object positions will not change throughout the construction algorithm. For the analysis of run-time complexity, we assume randomly chosen split positions for object partition, which leads to an overall run-time complexity of $O(n \log n)$ in the average case. The actual subdivision position is determined by the cost function, though.

Quality of Hierarchies

Figure 4.2 illustrates the resulting bounding box hierarchy when our method is applied to a real-world test scene (*aircraft*) containing more than 180,000 triangles. Even at a very low level of hierarchy (level 10) individual basic structural objects like seats and overhead compartments emerge. At level 12 details of the individual objects appear like armrests, backrests, or headrests. The total number of boxes at level 12 is less that 2^{12}, which is only about 2% of the total number of scene objects.

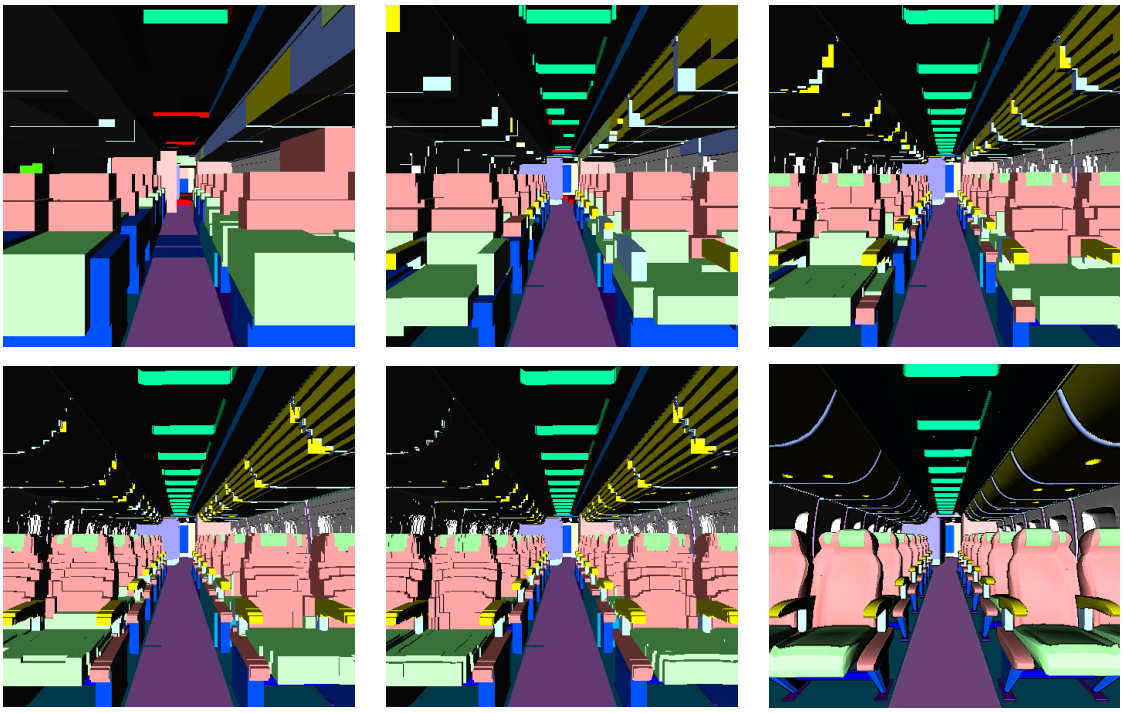


Figure 4.2 Visualization of bounding volumes of the *aircraft* test scene at tree levels 10, 12, 14, 16, and 18. The image at bottom right shows the Gouraud shaded original scene model.

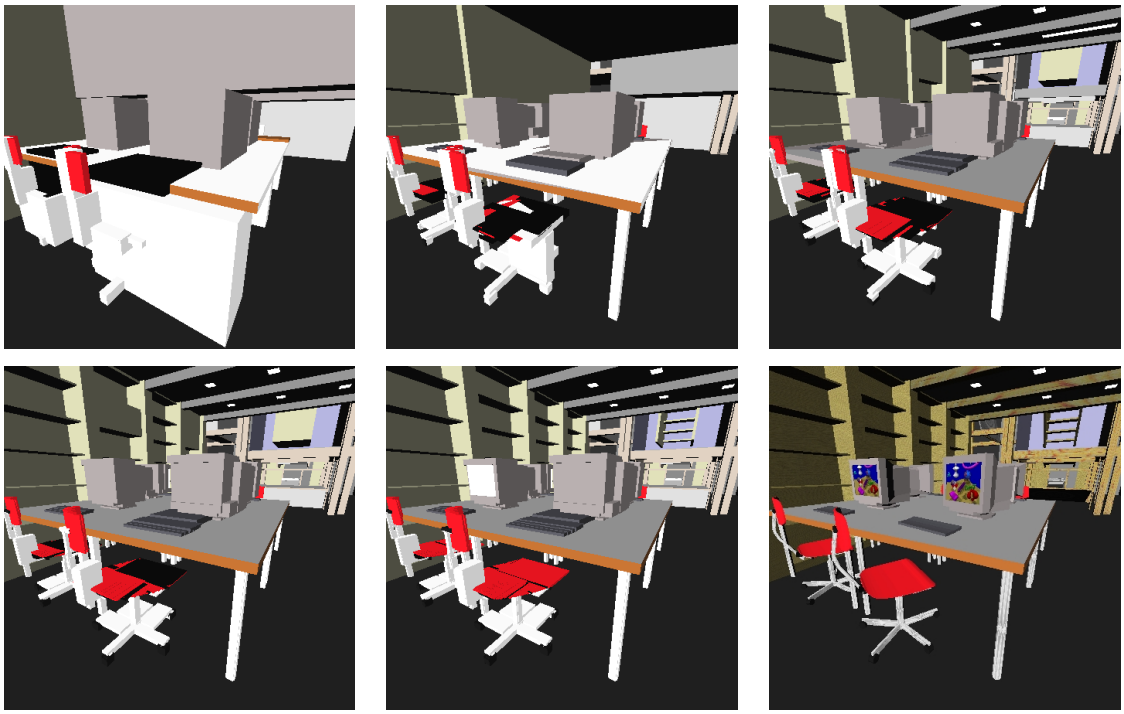


Figure 4.3 Visualization of bounding volumes of the *vrlab* test scene at tree levels 10, 12, 14, 16, and 18. The image at bottom right shows the Gouraud shaded original scene model.

Figure 4.3 shows the results for an architectural scene containing more that 36,000 polygons. Again, close fitting bounding volumes emerge at a relatively low level of the hierarchy. Please note that at level 12 details of the scene like chairs, tables, cubicles, monitors, and keyboard are detected at a very abstract but informing degree. At level 14 smaller details like a ladder to the second floor or book shelves emerge.

The clustering scheme even detects inhomogeneous detail at a very early level of hierarchy and is thus suitable for a vast range of scene arrangements.

4.2.3 Optimizing Ray Acceleration

Besides the quality of the cluster hierarchy the performance of the radiosity algorithm is mostly influenced by the efficiency of the underlying ray acceleration scheme. Although bounding volume hierarchies deliver acceptable performance in many cases, these schemes involve high tree traversal costs which can often be reduced by using hierarchical grid approaches. To answer visibility queries, we have presented a novel hybrid ray acceleration scheme [MF99] that combines the advantages of bounding volume hierarchies with uniform grid approaches. We have shown that the performance is at least equal to similar schemes like [CDP95] and [KS97]. In this chapter, we will only summarize the major subdivision algorithm and its results:

Given a bounding volume hierarchy, our goal is to detect inner scene nodes that are suitable base nodes for a local space subdivision. To achieve this goal, we recursively classify scene nodes based on the

- **surface area** of neighbor hierarchy nodes
- **volume** of neighbor hierarchy nodes
- **average size** of elementary objects below a hierarchy node

We classify an inner scene node as a suitable base node for a uniform space subdivision if the surface area and the volume of the node are not significantly larger than respective summed values of its child nodes. Also, we demand that the elementary objects below each child node have similar size to justify a locally uniform space subdivision for this sub-scene. The involved threshold constants were determined empirically.

Additionally, all scene nodes hold a counter representing the number of *uniform* classified sub-nodes which will be used in the actual space subdivision phase. In the next phase, uniform space subdivisions are built for sub-scenes marked in the previous step. This is efficiently achieved by recursively subdividing the bounding box of a scene node along the dominant coordinate axis. The node counter is used to determine the number of voxels or subdivisions. The available bounding volume hierarchy can be used to speed-up the voxel initialization significantly by applying hierarchical voxel membership tests.

Figure 4.4 shows an example of a resulting data structure. A scene is partitioned into two (possibly distant) subscenes using a bounding volume hierarchy. One subscene contains many subscenes uniformly distributed in space, thus we are building a uniform spatial subdivision to represent the subscene. Note that highly inhomogeneous detail contained in one of the resulting voxels, could again be modeled using a hierarchy of bounding volumes.

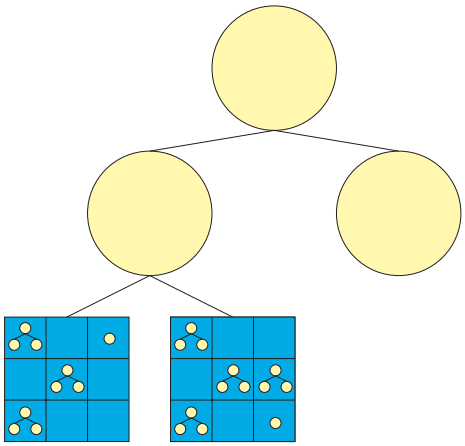


Figure 4.4 Combining bounding volume hierarchies and space subdivisions.

Construction Costs

We have shown in [MF99] that both, run-time complexity and space complexity of the space-subdivision approach is linear in the number of scene objects. Thus, the overall complexity of the cluster construction is dominated by the time to build bounding volume hierarchy. This is a remarkable fact, because it implies, that the whole scene voxelization can be done in a very short time, provided the scene hierarchy from Section 4.2.2 is available. We could compute the scene hierarchy *once*, save the result to an external storage medium, and perform a fast voxelization based on the precomputed scene hierarchy, whenever using the scene.

4.3 Radiosity with Optimized Clusters

4.3.1 Overview of the Implementation

Our radiosity implementation mainly follows the one presented in [GH96], where the authors propose using an error driven refinement technique to improve the efficiency of the algorithm. To decide whether a given link has to be subdivided, upper and lower error bounds on the energy transfer are compared. These error bounds are a combination of bounds on form factors, visibility and irradiance transfer involved in that particular interaction. During energy exchange bounds are gathered and propagated exactly like radiosity thus accurately reflecting the quality of the transfer. BF-refinement [HSA91], which is often used in standard hierarchical radiosity only uses the product of form factor and radiosity as an indicator for link refinement. The error driven approach leads to fewer links when compared to BF-refinement and it produces more subdivisions at the element level, thereby improving the detection of shadow boundaries.

The second important detail is the way how energy is transferred from and to clusters. Energy received by a cluster is directly deposited on the contained surfaces obeying their orientation with respect to the source. An iteration process that pushes the energy down the tree is started once the top level surfaces are reached. When energy is gathered from a cluster, surface orientation inside that cluster also plays an important role. To account for the non-diffuse re-

flection property of a surface cluster, the reflection of each contained surface is weighted by its orientation to the receiver.

Visibility calculations between patches or between patches and clusters are done via ray casting. Only if visibility between two clusters is needed, we use the approximative visibility approach based on a voxel grid with extinction coefficients [Sil95].

4.3.2 Using the Cluster Hierarchy

The algorithm described in Section 4.2.2 builds a binary tree containing a bounding volume hierarchy. If the algorithm detected regions of regular object densities inner nodes encapsulating these regions are specially marked. Inside these regular nodes, however, the bounding volume hierarchy still persists. This is important, because the algorithm might find regular regions quite early, thus probably resulting in no hierarchy at all. Because we want to use the same hierarchy for ray tracing (i.e. visibility checks) and for clustering we decided to combine both data structures. Ray tracing can benefit from regular regions whereas the radiosity algorithm needs the full hierarchy. Nodes with a regular structure are derived from standard inner nodes (our implementation is done in C++) and overload the method performing a ray intersection test. Instead of just propagating the test to their child nodes, regular nodes use a local voxel array created during the construction of the hierarchy.

When performing our first tests with this data structure applied to large models, we found that most time was spent during gathering energy over the links and propagating it through the hierarchy. The refinement steps consisting of form factor calculations, visibility checks, and subdivision of links however, only used a fraction of the gathering time. The reason for both observations lies in the nature of our hierarchy. The subdivision of links is very fast due to the branching factor of two in a binary tree. The visibility checks are performed either approximatively or based on ray casting which greatly benefits from the tight fitting bounding volumes and the local voxel arrays. The problem we encountered in the gathering step however, is the depth of the tree.

Optimizing the Directional Transfer

As mentioned above, propagation of energy in a cluster is performed by depositing energy directly to the inner surfaces. This means traversing the hierarchy below a cluster to enumerate all leaves and computing a dot product of each surface normal with the direction of transfer. Thus, performing this step for each link and at all levels of the hierarchy results in an excessive number of partial sweeps through the hierarchy. Because the same traversal has also to be done for the sending cluster, deep hierarchies degrade the performance of this process. To optimize the number of iterations on both sides (sender and receiver) our gathering step for clusters first loops over all links. While visiting each link, the sender's weighted radiosity is computed once and pushed on a stack together with the direction of transfer. After all links are visited, the receiving cluster's resembling surfaces are enumerated and the saved radiosities are accumulated on these children, again weighted by their orientation. This results in only two traversals for each link. This approach is similar to the α-links proposed by Smits et al. [SAG94]. In our implementation however, the direction of transfer for each individual contribution to the receiver's radiance is taken into account. Smits computes a single average value for the sender

and uses it to update all patches of the receiving cluster which can be wrong if most patches are oriented sideways regarding the direction of transfer. Using the technique described above a higher accuracy is obtained without increasing the complexity.

Efficiently Solving the System

The original hierarchical radiosity algorithm [HSA91] and subsequent clustering algorithms [Sil95, SAG94] split the solution process in two passes. In a gathering step the energy is transferred across the links, which is done for all links of all nodes. In a second pass the radiances must be swept to each object's parents and children to obtain the proper amount of energy at all levels of the hierarchy. The radiosity received by a parent node is pushed down the hierarchy to the children and on the way up the parents receive the area-weighted average radiosities from their children. The separation of a *gathering* and a *push-pull* pass corresponds to the Jacobi iteration, where the solution vector is only updated after a full iteration step [SP94]. To speed up the convergence rate and thereby minimizing the number of full hierarchy traversals Gauss-Seidel iteration should be used, which means that the entries of the solution vector are updated in place during an iteration. A combined gather and push-pull procedure is recursively called for each child node. At each element radiosity is gathered and pushed down the hierarchy. At the leaf level the emissivity is added and the radiosities are area-averaged on the upward pass [GH96]. Thus, energy can be propagated in a single sweep without an additional push-pull step as required for Jacobi iterations. We consider a gathering step to be complete when the maximum energy gain drops below some fixed threshold. For our test scenes this never required more than 3 full iterations for a single gathering step.

4.4 Results

To test the overall behavior and efficiency of our clustering algorithm we computed radiosity solutions for several large scenes containing up to 180,000 polygons (see Figure 4.5).

The scenes chosen cover a broad range of features that are typically not found in 'synthetic' test scenes. The *aircraft* is a highly tesselated model of the interior of an aircraft with mainly curved surfaces. *wichmann* is an architectural model of an office building containing moderately subdivided furniture and many large light sources. Most surfaces of the scene are textured. The *vrlab* scene has lower overall complexity than the former one but here the chairs are highly tesselated. The *atrium* scene has the lowest complexity of our test scenes, but it is rather irregular due to several trees on the ground floor.

For rendering times and additional statistics please refer to Table 4.6 (all tests were run on a 250MHz R10k with 2GB RAM). To measure the quality of the clustering algorithm, we have chosen the error threshold to not subdivide more than about 20% of the initial polygons. This ensures that the timings reflect the quality of the bounding volume hierarchy as well as the ray-tracing speed. Finer error thresholds shift the main computational effort towards ray tracing. Solutions of higher quality are presented in Figure 4.7.

4.5 Discussion

Our impression from these tests is that the algorithm in generally behaves very well. This confirms the results regarding the quality of our bounding volume hierarchy documented in Figures 4.2 and 4.3. Although we did not directly compare our algorithm to one of the clustering algorithms published before, two of our test scenes (*aircraft* and *vrlab*) have already been used in [HDSD99]. Comparing our rendering times at the given visual quality to the former results shows that using the same hierarchy for clustering and ray acceleration is a very promising approach.

Beyond these observations our algorithm closely matches several requirements found to be useful in clustering algorithms [HDSD99]: The creation time of the hierarchy is very efficient and tightly fitting bounding volumes are generated. These bounding volumes are stored in a binary tree, which allows for fast link refinement steps. Additionally, the pure ray-tracing speed of our bounding volume hierarchy combined with voxel grids has been shown to be superior to most other approaches [MF99].

The reliable detection of almost arbitrary scene details, even at moderate hierarchy depths opens several perspectives for future work. Rapid prototyping applications using the tight bounding boxes as an approximation of a scene could improve the efficiency of various algorithms. For instance, a radiosity computation could be performed on only a fraction of the number of the original scene elements. Projecting the results obtained by this method onto the real scene polygons (i.e. using texture mapping hardware) would deliver high-speed approximative solutions, applicable to dynamic environments.

Figure 4.5 Renderings of the four test scenes. From left to right and from top to bottom: *aircraft* (Light-Work Design Ltd.), *wichmann*, *atrium*, *vrlab*

	aircraft	*wichmann*	*atrium*	*vrlab*
number of polygons	183,114	59,844	13,558	36,337
max. hierarchy depth	26	27	25	27
hierarchy creation [min:sec]	0:53	0:22	0:03	0:12
rendering time [min:sec]	14:37	7:01	3:29	6:55

Figure 4.6 Rendering times and statistics for the rendered test scenes.

4.7.a) interior view

4.7.b) view from outside

Figure 4.7 High quality renderings of the scene *wichmann* (47 min)

CHAPTER 5

Generation of Decomposition Hierarchies for Efficient Occlusion Culling of Large Polygonal Models

Abstract

Efficient handling of large polygonal scenes has always been a challenging task and in recent years, view-frustum and occlusion culling have drawn a lot of attention for accomplishing this goal. The problem of how to efficiently organize such scenes for fast image synthesis is widely neglected, although the answer heavily affects the overall performance. In this chapter, we present three adapted algorithms for efficient scene decomposition and compare those with another already available algorithm for decomposing general polygonal models into a hierarchy of sub-models for an occlusion culling application. While the latter is available as a commercial product, the three other approaches introduce new algorithms for scene decomposition which achieve significant better results.

5.1 Introduction

Hierarchical methods are crucial for the efficient handling of large computer graphics scenes. Several approaches address the problem of generating scene hierarchies, such as subdivision surfaces [ZSD+00], multi-resolution frameworks [Hop96], scene databases [ZMHH97], or regular decomposition schemes such as BSP-trees [FKN80] or octrees [Sam94]. An important application field of scene organization is view-frustum and occlusion culling, which specifically requires appropriate decomposition schemes. However, most available approaches lack either flexibility or efficient handling.

5.1.1 Related Work

Several previous papers on visibility and occlusion culling touch the topic of scene organization. Generally, we observed that decomposition schemes for arbitrary polygonal models are difficult to derive. Hong et al. [HMK+97] used a technique which decomposes a CT-based colon volume dataset along its skeleton. The size of the different decomposition entities depends on how

many voxels belong to this entity. While this scheme produced good results for a tube-like colon model, it is not efficient for general models.

Snyder and Lengyel [SL98] proposed that the designer of the scene needs to provide the scene organization. Similarly, Zhang et al. used a pre-defined scene database [ZMHH97, HMC+97]. Models built in large Computer-Aided-Design (CAD) systems might already include appropriate hierarchical organization information, due to the design process which uses hierarchical notions like grouping and replication. Besides approaches which use a building floor plan for this purpose [ARB90, Air90, TS91, LG95], no other methods for deriving decomposition hierarchies from CAD models are known to us.

A more general approach is to organize a polygonal model into regular spatial decomposition schemes, such as BSP-trees [FKN80, Nay92, Gre96] or Octrees [Gre95, GKM93, CT96, CT97, SG96]. While these decomposition schemes produce good results on polygonal models extracted by the Marching Cubes algorithm from uniform grid volume datasets — which provide a "natural" decomposition on Marching Cubes cell base —, these schemes run into numerous problems on general models. If a polygon of the model lies across a decomposition boundary, it must be either split into several parts, in order to produce a disjunct representation of the bounding entities, or handled in another special way. Splitting polygons however, can increase the number of small and narrow polygons tremendously.

Significant work on model organization has been published in the field of collision detection. Methods based on oriented bounding boxes (OBB) were explored by Gottschalk et al. [GLM96]. A bottom-up approach for the construction of a model hierarchy is suggested in [BCG+96] in which nodes representing small parts of the geometry are "merged" into higher hierarchy nodes. A similar approach is used in [KMM+98]. Related model organization strategies are also known from global illumination.

In [BMH99], the spatialization functionality of SGI's OpenGL Optimizer package [SGI97] was used to generates scene hierarchies automatically. However, these decomposition hierarchies needed to be tuned manually in order to get sufficient performance and motivated the work described in this chapter.

5.2 Occlusion Culling

Generally, a polygonal scene can be decomposed into a hierarchical or non-hierarchical decomposition, denoted *scene tree* or *scene graph*. Each part of it is a *scene entity* which can be either be a *scene node* or a *geometry node*. The latter contains geometry of the actual scene.

The decomposition quality of the four presented approaches is evaluated with a standard hierarchical occlusion culling algorithm [BS99] based on the HP-Occlusion-Culling-Flag [SOG98] and a basic view-frustum culling algorithm [BMH99]. A given scene tree hierarchy is processed in top-down, left-right order. Initially, the actual polygons of the geometry node closest to the eye is rendered into the empty framebuffer. Then, each scene entity of the tree is tested for intersection with the view-frustum (view-frustum culling). Thereafter, all the remaining scene entities are depth-sorted according to their associated bounding boxes and tested for occlusion using the HP flag. All further rendering is performed in an interleaved fashion. First, bounding box of the next closest scene entity is rendered in a special occlusion mode which does not alter the framebuffer. If this bounding box would not have any contribution to the framebuffer, the

HP Occlusion Culling Flag is set FALSE by the graphics hardware and consequently this scene entity is culled. Otherwise (flag is set TRUE), the child scene entities are processed unless the entity itself is a geometry node; in this case, the polygons are rendered in standard render mode.

5.3 Generating Decomposition Hierarchies

Generating hierarchical decomposition of very large models can be done in numerous ways. In this section, we will present three research algorithms and one commercial tool which generate a decomposition hierarchy starting from given models: *D-BVD*, *p-HBVO*, *ORSD*, and *SGI*. The latter is part of SGI's OpenGL Optimizer toolkit (opoptimize), while the other three novel algorithms have been developed and implemented by the authors. All approaches decompose general polygonal models, whereas the octree-based ORSD only decomposes regular grid datasets.

5.3.1 Dimension-oriented Bounding Volume Decomposition (D-BVD)

The goal of the dimension oriented D-BVD decomposition algorithm is to generate evenly-sized, cube-shaped bounding boxes, hence minimizing the area of the screen projection of these bounding boxes. This goal is approached by splitting the bounding boxes multiple times in the largest dimension. The size of the bounding boxes and the associated sub-models is controlled by user-specified parameters, such as the minimal number of polygons.

Starting with the root scene entity – which contains the whole model – the associated bounding volume (bv) is split n times along its largest dimension such that each fraction is approximately of the same size as the second largest dimension.

$$n = \frac{\text{largest bv dimension}}{\text{second largest bv dimension}} \tag{5.1}$$

This process continues recursively until the termination criteria are met. These criteria give lower bounds for the number of polygons of the scene entities or the size of the dimensions of the associated bounding boxes, in order to avoid undersized decomposition entities which increase the occlusion culling overhead without improving the cull rate sufficiently.

Occasionally, the split-operation of the decomposition process splits a polygon which lies across the decomposition boundary into two new polygons (this is usually not the case for uniform grid datasets). This can tremendously increase the number of polygons and frequently, these polygons are small and narrow, thus resulting in numerical problems. To compensate this, we apply two techniques. First, the decomposition plane is moved along the decomposition dimension to reduce the number of additional polygons. The direction and value of the movement is controlled by user-specified parameters. Nevertheless, very large polygons cannot be handled by this technique, because they cover several high-level scene entities (i.e., a single polygon representing the floor of a factory). To account for this, we use a second technique, where those polygons are *pulled up* into a geometry node located at the appropriate level in the scene tree.

In general, this algorithm can handle all types of polygonal scenes without producing significantly more polygons. It optimizes shape and size of the scene entities, hence their bounding

boxes. However, the polygon load is not evenly distributed to the scene entities, possibly resulting in a less balanced scene tree.

5.3.2 Polygon-based Hierarchical Bounding Volume Optimization (p-HBVO)

The polygon-based Hierarchical-Bounding-Volume-Optimization (*p-HBVO*) method decomposes recursively a set of polygons into two sets. Instead of arbitrarily selecting possible decomposition planes, these planes are given by the barycenter of each polygon (triangle). By evaluating a cost-function, an optimal decomposition plane is established. At each decomposition level, the individual polygons are assigned to exactly one scene entity of that level resulting in possibly partially overlapping bounding boxes but in no additional polygons.

Starting from the root node, at each decomposition step, we sort the polygons along all coordinate axes, where the barycenter of each polygon serves as sorting key. Based on these three ordered lists, we evaluate the potential decomposition planes along each axis for each entry in the respective list by splitting the sorted list of polygons into a *left* and *right* part. In contrast to pre-defined decomposition planes of the median cut scheme [KK86b], we evaluate for each possible decomposition plane – defined by the entries in the lists – a cost function which approximates the costs of rendering the polygons of one of the two scene entities, generated by the respective decomposition plane. By minimizing this cost-function over all possible decomposition planes, we obtain an optimal decomposition plane which generates two new scene entities; one contains all *left* polygons, the other one contains all *right* polygons. The decomposition process terminates when either the number of polygons, or the decomposition depth exceeds one of the two pre-defined parameters: *Max_Triangles_Per_Decomposition_Entity* or *Max_Decomposition_Depth*. These parameters are specified by the user and supplied at the start of the decomposition process.

In most cases, our cost function is identical to one which has already been successfully applied in ray tracing environments [MF99]. Adopting this cost function is possible since the objective is the same; both algorithms traverse the scene graph in a similar way to determine visibility. The costs of a scene entity H, with left H_l and right H_r children, is given by:

$$C_H(axis) \quad = \quad \frac{S(H_l)}{S(H)} \cdot |H_l| + \frac{S(H_r)}{S(H)} \cdot |H_r|$$

where

- $|H|$ is the number of polygons within hierarchy H,
- $S(H)$ the surface area of the bounding box associated to sub-scene H, and
- $axis \in \{X, Y, Z\}$.

Overall, this algorithm generates well balanced scene trees with respect to their polygon load. Furthermore, polygons of individual objects are detected and clustered together (see Section 5.4, city dataset). The highest culling performance was achieved with finer decompositions, which usually require more occlusion culling tests, resulting in higher culling costs. If these culling costs are not compensated by lower rendering costs, it results in an overall lower rendering rate (see ventricular system in Table 5.1).

5.3.3 Octree-based Regular Space Decomposition (ORSD)

As mentioned earlier, regular decompositions are well suited for uniform grid datasets, such as generated by MRI or CT scanners (i.e., ventricle dataset). Uniform grid datasets consist of a set of sample values (voxels), arranged on an uniform grid. A cube of eight neighboring voxels is called a cell, where cells with a non-empty intersection with the selected isosurface are called relevant cells.

The Octree-based Regular Space Decomposition method (ORSD) exploits these natural decomposition borders to generate a non-polygon splitting model decomposition. In contrast to the other approaches, ORSD uses a cell-based (voxel-based) evaluation criterion, where the number of relevant cells (relevant cell load or RCL) controls the decomposition process. This criterion is only a rough approximation of the actual number of extracted polygons, considering that each relevant cell represents between one and five triangles. In our experiences however, RCL turned out to be sufficiently accurate.

After the construction of the entire octree, the RCL of each octant is already calculated. Subsequently, the octree is traversed recursively, starting with the superblock. If the RCL is above a user-specified threshold, the block is classified as a scene node, thus being further decomposed into its child blocks. Otherwise, a geometry node is instantiated with the corresponding polygons (associated relevant cells).

Overall, ORSD is a simple but efficient decomposition scheme which generates an adequate polygon load balance and bounding box sizes. As shown in the results (see Section 5.4), the indirect evaluation method (RCL instead of number of polygons) does not adversely affect the occlusion culling performance. However, the implemented version of ORSD is limited to regular grids.

5.3.4 SGI's opoptimize (SGI)

SGI's OpenGL Optimizer is a C++ toolkit for CAD applications that provides scene graph functionality for handling and visualizing large polygonal scenes. It includes mechanisms for the decomposition of model databases as well as for tessellation, simplification, and others.

Opoptimize (depicted in the following sections as "SGI"), which is part of the toolkit, provides functionality for the decomposition of model databases. The decomposition method realized in SGI is similar to the construction of an octree; each geometry set is split into eight equally sized sets. This process is repeated recursively, until a certain threshold criteria for the iterated decompositions is reached.

Octree-based spatial decomposition is a simple and efficient decomposition scheme. However, the SGI decomposition mechanism decomposes space not by simply bisecting edges of a cube, as in an octree, but by choosing decomposition planes such that the rendering loads of the resulting parts are similar. As a result, the amount of geometry after each decomposition on each side of the decomposition plane is approximately the same. Polygons which are split due to the decomposition are distributed to the respective geometry nodes. The main parameters that can be used to control the decomposition are hints for the lowest and highest amount of triangles (tri_{min}, tri_{max}) in each geometry node at the leaf-level of the decomposition hierarchy. However, the decomposition algorithm only tries to meet these criteria but is not bound to it.

In general, SGI generates decomposition hierarchies with a well balanced polygon load.

However, the bounding boxes of the scene nodes are less suited for occlusion culling applications, because the cost function determining the subdivision is obviously not optimized with respect to the volume of the bounding boxes. We observed that the right-most branch of the scene tree frequently contained large subsets (bounding box volume size) of the model, even in the lower tree levels.

5.4 Results

Ventricle				Cathedral dataset			City dataset		
pHBVO	D-BVD	ORSD	SGI	pHBVO	D-BVD	SGI	pHBVO	D-BVD	SGI
80	41	6	29	9	59	51	2722	266	420
81	41	28	36	10	67	52	2723	495	420
0.008	0.007	0.003	0.004	0.002	0.008	0.01	0.02	0.01	0.01
0.03	0.02	0.01	0.016	0.004	0.032	0.03	0.05	0.03	0.03
0.05	0.06	0.06	0.06	0.136	0.124	0.14	0.01	0.08	0.08
16.0	19.7	22.4	19.7	30.0	25.9	30.1	0.1	4.1	3.6
12.3	13.4	15.3	13.6	12.4	8.8	7.8	14.0	10.9	11.8
29.4	21.3	17.0	21.1	33.1	25.9	22.2	47.0	50.7	50.4
5.5	5.3	5.3	5.3	12.4	10.8	9.5	1.0	1.4	1.4

Table 5.1 Walk-through Measurements: rows from top to bottom: #Scene nodes. #Leaf nodes. View-frustum culling time [s]. Occlusion culling time [s]. Rendering time [s]. Rendering rate [% of original polygon count]. Frame rate [fps]. Rendering rate [%], view-frustum culling only. Frame rate [%], view-frustum culling only.

In this section, we discuss the efficiency of our three novel algorithms and opoptimize of SGI's OpenGL Optimizer (SGI) with respect to their occlusion culling-based render performance. All measurements are performed on a HP B180/fx4 graphics workstation. The three different polygonal datasets (see Table 5.2) represent typical scenarios of different application areas. During our evaluation, we measured the time spent for view-frustum culling, occlusion culling, and rendering of the not occluded geometry of different decomposition granularities. For the evaluation, the decomposition with the best culling-based render performance was selected at the averaged results are shown in Table 5.1. In addition, frame rates of walk-throughs of the datasets and the respective render rate, which mirrors the percentage of the geometry of the scene which was rendered (this is reciprocal to the cull rate) are presented, see Figure 5.2.

Ventricle Dataset

The first dataset is a polygonal model of the ventricular system of the human brain extracted from a MRI scan, see Figure 5.3(a,b) and Table 5.2. We explore the dataset by moving through the lower part (Cisterna Magna) of the polygonal model. Most of the model structures throughout the walk-through are located within the view-frustum, while the structures with the largest

Dataset	Grid Type/ Source	#Triangles	FR
Ventricular System	Uniform/ MRI	270,882	4.6
Cathedral	Unstructured/ CAD	416,763	3.8
City	Unstructured/ Modeler	1,408,152	0.9

Table 5.2 Models; as gold standard for the speed-up due to occlusion culling, we show the frame rate for the datasets without any culling mechanism (FR).

number of polygons (located in the upper part or lateral ventricles) were not visible due to occlusion. All polygons of this model are aligned on the uniform cell grid and are approximately of the same size. All three adapted algorithms were able to detect this “natural” decomposition boundaries; only SGI generated approximately 15% additional polygons due to splitting operation between the grid points.

Figure 5.2 show frame rate (a) and render rate (b) of the four evaluated algorithms and Table 5.1 shows the averaged time measurements. The most interesting detail is the low amount of time consumed by view-frustum and occlusion culling by ORSD, due to its coarse decomposition. The render rate of p-HBVO and D-BVD was approximately 25% better than the render rate of the ORSD approach. However, the finer decomposition introduced additional culling costs twice as much as for ORSD, resulting in a lower frame rate.

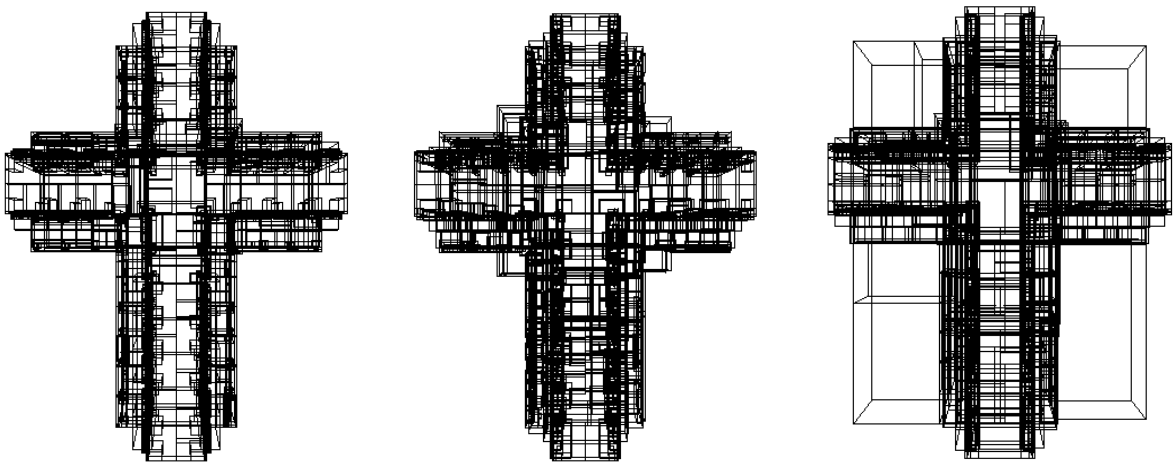
Cathedral Dataset

This dataset represents the interior of a gothic cathedral, designed with a CAD system (see Fig. 5.3(c,d) and Table 5.2). Occlusion is limited to small parts of the model, because a large share of the polygons are visible from most view points within the model. Figure 5.1 shows three fine decompositions of the cathedral model. Especially the p-HBVO approach (a) adapts very nicely to the structures of the model, such as pillars and arcs. In contrast, the decomposition generated by SGI (c) introduces very large bounding boxes, which do not adapt properly to the actual geometry.

The p-HBVO approach performed best on this dataset (see Figure 5.2(c,d)) which is due to the low culling costs, compared to SGI and D-BVD (see Table 5.1). The bounding boxes of D-BVD and SGI are not really suited for occlusion culling; especially their occlusion culling time is significantly higher compared to p-HBVO, thus severely reducing the frame rate. Consequently, view-frustum culling only is faster than occlusion culling for those approaches, while it is basically as fast as occlusion culling based on the p-HBVO hierarchy (see Table 5.1).

City Dataset

The city dataset is an artificial model of 400 basic building models with some interior, which contains most of the polygonal complexity, see Fig. 5.3(e,f) and Table 5.2. Consequently, most



(a) (b) (c)

Figure 5.1 Cathedral dataset – decomposed by (a) p-HBVO, (b) D-BVD, and (c) SGI; the arts and pillars of the cathedral are well detected by p-HBVO and D-BVD. SGI only used a regular spatial decomposition.

of the polygons of this model are occluded. However, only the p-HBVO approach was able to decompose all the interior into individual scene entities, hence resulting in a large number of nodes, a very low render rates, and high occlusion culling costs which were more than compensated by the small amount of potentially visible geometry (see Table 5.1 and Figure 5.2(e,f)). In contrast, the D-BVD approach did not detect the interior objects. The optimal computed decomposition was of coarser granularity (10% of the scene entities of p-HBVO), resulting in less time spent for culling (vfc and occ), but a higher render rate of 41 times larger than the render rate of p-HBVO (see Table 5.1). Similar, the SGI approach did also not detect the interior objects. It used a coarse granular decomposition, which consequently increased the rendering time in comparison to p-HBVO.

Summary

Overall, two of our three adapted model organization approaches were able to generate decompositions with faster rendering due to higher cull performance. This was achieved by reducing culling costs or by reducing the render rate of the dataset. On uniform grid datasets, the basic ORSD approach produced a model decomposition which performed best, mostly due to the low time spent to establish occlusion or non-occlusion.

Generally, we observed that models with high occlusion do not require very fine decomposition (ventricle dataset, D-BVD vs. ORSD). On the other hand, a fine decomposition pays off if interior (thus completely occluded) objects are clustered in a scene entity (city dataset, p-HBVO vs. SGI). In contrast, models with low occlusion (cathedral dataset) can benefit from finer decompositions if the culling costs loss is only a fraction of the rendering costs gain (see city dataset, p-HBVO). However, this was not true of the cathedral dataset (D-BVD).

Note that the p-HBVO approach builds a binary scene tree. This usually results in deeper trees, hence more intermediate scene entities. This increases the time spent for occlusion culling

significantly. Once this binary tree was re-build into a quad tree representation, we accomplished a frame rate increase by approximately 20%.

To summarize, we always achieved a speed-up due to occlusion culling-based rendering. Especially with the city dataset, we accomplished a speed-up of 15.6 after culling of 99.9% of the model geometry with the p-HBVO algorithm. The other approaches obtained a similar speed-up, while culling less polygons.
On the ventricle dataset, the ORSD approach accomplished the best results; 77.6% of the geometry were culled, due to view-frustum and occlusion culling. This culling performance resulted in a frame rate speed-up of 3.3.

5.5 Conclusion and Future Work

Four different approaches for hierarchical decompositions of large polygonal models were presented and discussed. One of them is a commercially available product (SGI's OpenGL Optimizer), while the other approaches are still research projects. Overall, the former approaches achieved similar (D-BVD) or better performance — evaluated with an occlusion culling application — than the tool of SGI's OpenGL Optimizer. It turned out that a simple octree-based scheme (ORSD) suits very well to uniform grid datasets; bounding box size as much as polygonal load balance were handled well. However, this scheme does not work on unstructured grid datasets because of the missing "natural" decomposition on cell-base. In particular the p-HBVO approach performed well on all datasets. Whereas the D-BVD approach optimized the size and shape of the bounding boxes, the p-HBVO approach was able to generate good bounding boxes, while also balancing the polygon load.

So far, only spatial coherence information is exploited in order to combine raw triangles into scene entities. If neighborhood connectivity information –, i.e., semantic information which describes what kind of object should be combined (like the roof of a house in the city dataset) – is used, we expect decompositions which perform better than the current ones. This will be a major future research focus.

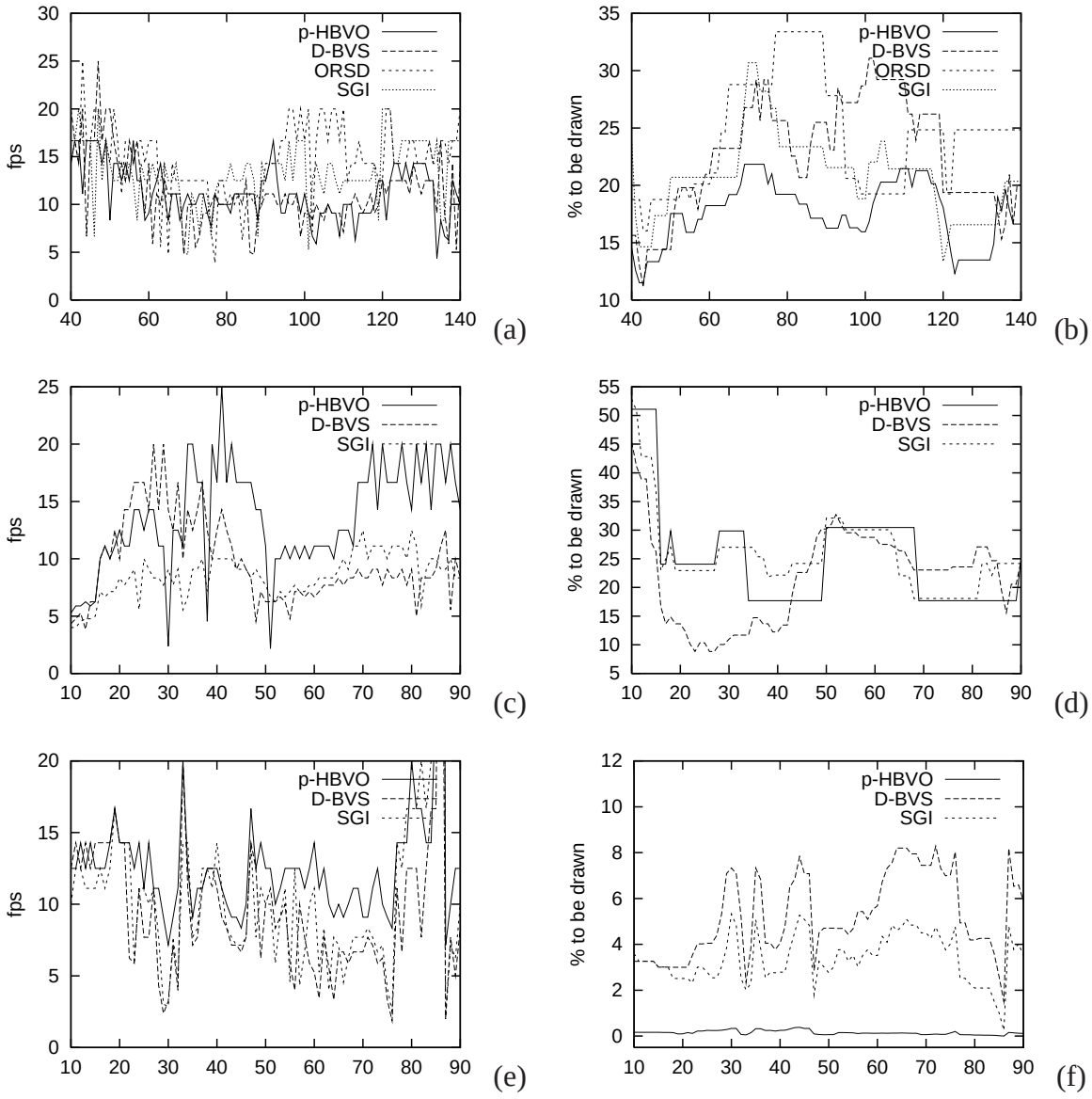


Figure 5.2 Decomposition results: Left column shows frame-rate and right column shows final geometry drawn. (a,b) Ventricle dataset; window of frames 40 to 140. (c,d) Cathedral dataset; all frames of the path are shown. (e,f) City dataset; window of frames 10 to 90 (out of 200).

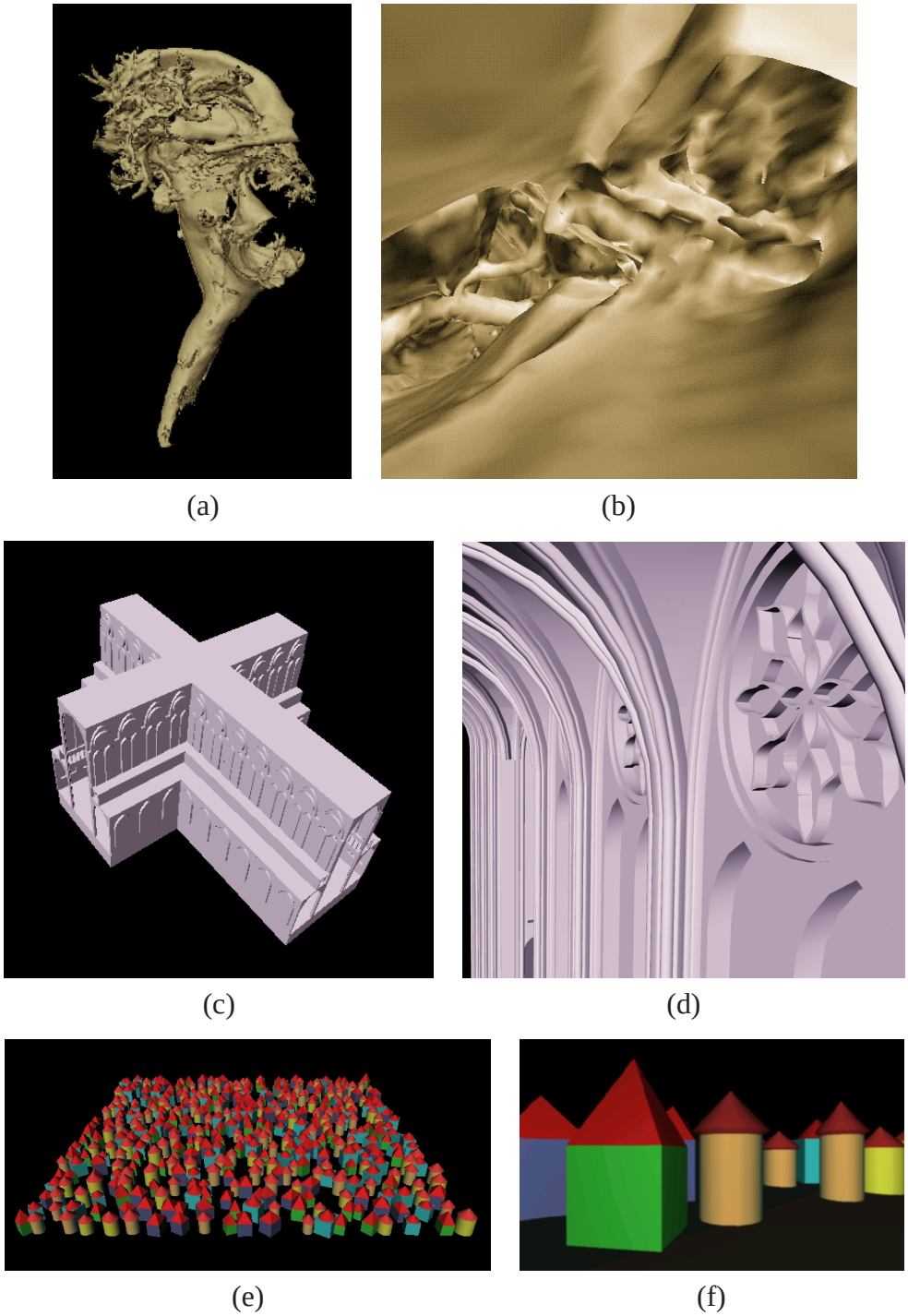


Figure 5.3 Ventricle dataset: (a) Overview, (b) Inside view. Cathedral dataset: (c) Overview, (d) Inside view. City dataset: (e) Overview, (f) Inside view.

CHAPTER 6

An Adaptive Hierarchical Occlusion Culling Algorithm for Interactive Large Model Visualization

Abstract

This chapter describes a new occlusion culling technique for efficient visibility determination in real-time environments. There are two major contributions of our work. First, we present an efficient algorithm to determine a set of potential occluders by sampling the virtual environment by sending rays from the camera into the view frustum. Thus, the set of potential occluders is always a subset of actually visible objects. We show how we can reduce the number of occlusion tests based on the fact that we use only visible occluders. Second, we present an algorithm that reduces the number of unsuccessful tests for occlusion significantly. We control the activation and de-activation of potential occluders for actual testing based on previous testing results. This control reduces the number of occlusion tests by a factor of 1.5-10 depending on the underlying scene geometry compared to testing against all potential occluders. This results in a significant increase of the frame rate. Our algorithms do not make any assumptions on how the actual occlusion test for a single object against a different object is implemented. Rather, we present a high level frame-work to control the occlusion tests. We exploit frame-to-frame coherence and the underlying techniques can easily be adapted to dynamic environments with moving objects.

6.1 Introduction

Over the last few years the complexity of 3D models used for real-time visualization has grown significantly. Despite the fast increase of raw rendering power that has lead to extremely powerful graphics processor capable of handling millions of polygons per second, this growth was beaten by the explosion of the size of architectural and scientific data sets. These huge data sets can hardly be displayed interactively with acceptable speed even with most modern hardware available. We can expect that this trend will continue in the future, since there does not seem to be a natural upper limit in model complexity to realize more and more realistic artificial environments.

To adress the discrepancy between available raw rendering power and model size, several acceleration techniques have been developed in the past:

Level-of-detail (LOD) techniques internally represent the 3D model at multiple resolutions and select for display an appropriate (often low-resolution) representation depending of the current view parameters. Simple LOD schemes only hold a fixed number of resolutions while improved techniques such as *progressive meshes* allow for smooth transitions between arbitrary levels [Hop96].

Impostors approximate complex objects by texture mapped polygons. Such images are typically generated dynamically and should be re-used over several subsequent frames to achieve a performance gain. Improved techniques make use of object hierarchies [SS96] and include depth information into the image (*nailboards*) to extend an imposter's lifetime [Sch97].

Culling techniques reduce the geometric complexity of the model by removing invisible parts before sending them to the graphics pipeline. *View frustum culling* eliminates objects outside the field-of-view of a spectator and *occlusion culling* algorithms detect geometry occluded by other objects. *Portal culling* [TS91]techniques discretize object space into cells and assign potential visible sets (PVS) of objects to each of the cells. PVS computation is mostly done in a computationally expensive pre-process, but recently a dynamic PVS construction scheme was presented in [WWS01].

Overview

This chapter will present speed-up techniques that improve both view frustum and occlusion culling algorithms by reducing the number of visibility tests performed during scene traversal. Section 6.2 describes occlusion culling algorithms that rely on an underlying hierarchical scene description. We analyse this class of algorithms in detail to identify situations where we can avoid useless visibility tests. Section 6.3 presents our scheme to dynamically select potential occluders from the scene of objects. An improved occlusion culling control scheme which activates and de-activates certain occlusion tests based on statistical results collected from previous frames is presented in Section 6.4. We compare our controlled occlusion culling framework to a standard implementation in Section 6.5 and conclude in Section 6.6.

6.2 Hierarchical Occlusion Culling

Many occlusion culling schemes rely on a hierarchical organisation of scene elements to allow for visibility checks at multiple levels of detail. Figure 6.1 shows an example distribution of objects 1 to 6 in a scene, Figure 6.2 shows a bounding volume hierarchy derived from these objects. Leafs nodes in this hierarchy represent elementary object, inner nodes represent bounding volumes enclosing their sub-scenes completely.

6.2.1 Object Hierarchy Generation

For hierarchy construction we use a top-down approach which was previously presented in [MF99] and [MSF99]. Basically, a cost function is minimized which approximates the expected costs to render the scene. The costs of rendering an inner scene node are approximated by

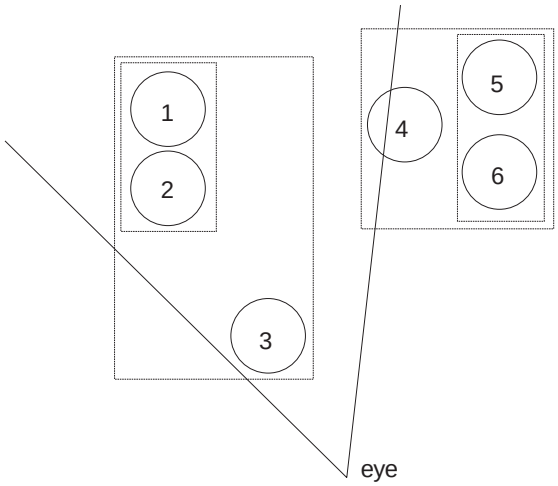


Figure 6.1 Sample scene geometry. Hierarchical bounding volumes are shown as boxes around objects 1 to 6. Objects 1 to 4 are located within the view frustum originating from the eye position.

summing up the individual costs of all of its sub-nodes times the probability of descending that sub-tree. The probability is approximated by the surface area of the sub-node's bounding volume and the costs are approximated by the surface area of primitives within this subtree. This results in very tight fitting (*natural*) bounding volumes as we have shown in [MSF99].

In a pre-processing step, primitives are sorted along the main coordinate axes. Then, for each coordinate axis, n primitives are binary partioned into a *left* and a *right* subsets containing at least 1 primitive. Each of the resulting $n-1$ partitions is evaluated using the previously mentioned cost function. The partition that minimizes the costs over all subdivision positions and all coordinate axes is chosen for subdivision and the algorithm continues recursively. Recursion is stopped when a predefined machine-specific minimum is reached. For our experiments we stopped subdivision when a set of less then 100 elementary objects resulted from subdivision. This number was found as the empirically determined limit where raw rendering was faster that testing for visibility plus rendering.

6.2.2 Tree Traversal

The occlusion algorithm which renders the whole scene works recursively and starts at the root of the hierarchy with a depth first traversal. We will concentrate on conservative culling techniques, which means that we will only remove an object from further processing if we are completely sure that the object will be invisible in the final output. The set of potential occluders is initiallly assumed to be empty and will be updated throughout the algorithm. Depending on the current type of the node, different actions take place, as follows.

Rendering Objects

If the current node is a leaf, the contained primitives are passed to the graphics pipeline and added to the set of potential occluders.

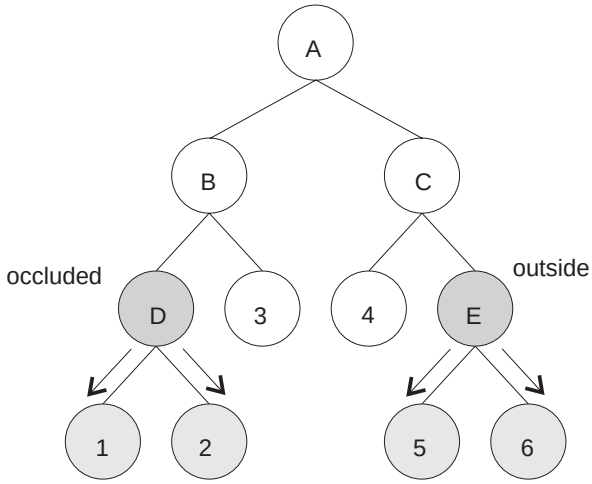


Figure 6.2 Knowing that an bounding volume in the object hierarchy is invisible from the current view, we can conclude that all sub-scene must also be invisible.

View Frustum Test

If the current node is an inner node of the scene hierarchy, the bounding volume of this node is tested against the current view frustum. If it is outside the current view frustum, no object within this bounding volume can be visible. The whole subtree may be pruned, no object within this subtree must be sent to the graphics pipeline. In the example of Figure 6.2 node E lies completey outside the view frustum, thus nodes 5 and 6 are not visible. If the current node is at least partially visible, the algorithm performs the occlusion test. If the current node is completely within the frustum further view frustum tests below this node may be omitted, since we can easily conclude that all objects must be inside the view frustum.

Occlusion Test

Next, the occlusion test is performed. The occlusion test determines if the bounding box of the current node is completely occluded from objects included in the set of potential occluders.

There are a lot of algorithms available to implement the occlusion test. Most of them work in image space and find out if rendering of the bounding volume would leave the frame buffer unchanged [GKM93, ZMHH97, BMH99, SOG98]. In this case we can conclude that the bounding box is occluded by previously rendered objects. In the example of Figure 6.2 node D could only be detected as occluded if object 3 was previously added to the set of potential occluders. This makes clear that objects should be rendered from front to back to find potential occluders as early as possible. For this reason we sort the sons of the current node with increasing distance from the eye point before descending the scene hierarchy . Since we have a binary tree when we use the algorithm from Section 6.2.1 for hierarchy generation, this sorting can be implemented easily as simple pointer swap operations.

6.2.3 Analysis

Most culling algorithms work similarly to the one described in Section 6.2.2. Although often a significant boost in rendering speed is achieved there is still the problem that every culling algorithm has some computational overhead compared to raw polygon rendering. Especially, occlusion tests resulting in "*potentially visible*" are quite disappointing, since we have wasted computational resources without any gain. In a worst case scenario where only few objects are occluded by others, we can even expect a performance degregation compared to raw rendering. Ideally, one should only perform visibility tests, if the result is *invisible* – at least with some high probability. Section 6.4 will present a framework that eliminates most of these unwanted tests.

Before we describe this occlusion culling control scheme, we will present a further optimization that deals with the rare case of recognizing a node as "*definitively visible*". If we know in advance that some object is visible from the view point, we can conclude that all bounding volumes on the path from the root of the hierarchy to this object must also be visible. At these nodes the visibility test can thus be omitted. E.g., if we knew that nodes 3 and 4 were visible, we may omit visibility test at nodes B, C, and A (Figure 6.3).

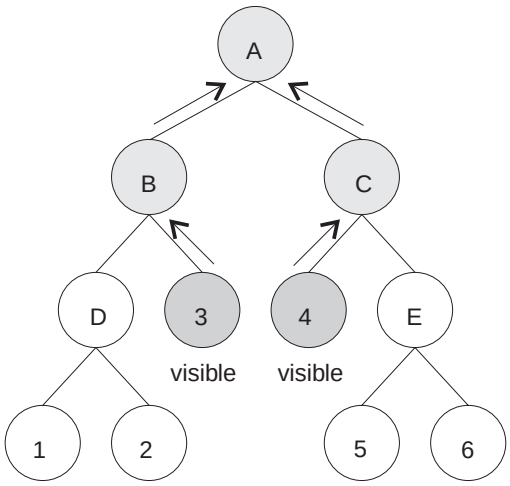


Figure 6.3 Knowing that certain primitives are visible from the current view point, we can conclude that all nodes on the path from this node to the root in the associated bounding volume hierarchy must also be visible.

6.3 Dynamic Occluder Selection

This section presents an optimization on the occlusion culling algorithm presented in Section 6.2. The basic idea is to find objects that are visible from the view point. These objects are added into the set of potential occluders before the hierarchy traversal starts up from the root. The choice of visible objects as occluders is promising, since they tend to hide many other objects in complex scenes. More importantly, the knowledge of visibility from a certain view, allows the propagation of this information up the bounding volume hierarchy which avoids all

occlusion or view frustum tests along the path from the root to the visible object. Thus, from knowledge of a small set of visible objects we eliminate many nodes close the root from testing. Since nodes near the root of the tree tend to have a rather large (projected) bounding volume, we can avoid many costy visibility tests, which would yield a "*partially visible*".

6.3.1 Visibility Sampling

The problem of identifying definitely visible objects to add to the set of occluders is rarely recognized in culling literature. The reason for this is that the visibility of a certain primitive in a classical z-buffered visualization environment is typically only known after all primitives have been rendered. Because we only need to find some visible objects to eliminate top level visibility tests, we simply sample the image space to identify visible objects at certain pixel coordinates. This could be done by using a z-buffer rendering into a 1x1 image buffer, but we have decided to use a ray casting approach since we can utilize otherwise unused CPU power resources.

The selection of sample positions is done using a quadtree like approach, the subdivision stops if a predefined number of samples is reached. Assuming uniform object distribution we assume that roughly $n^{\frac{2}{3}}$ objects are visible after projection if n is the number of leaves in the object hierarchy. The distribution of samples is adapted to the success of the sampling process (Figure 6.4). This is implemented as follows. A queue holds the sampling areas. Initially, the whole image area is added to this queue. While the queue is not empty and the maximum number of samples is not tested, an area element is taken from the queue and a random subdivision position x is chosen (stratified sampling). Next, the visibility test is performed at this sample. If no visible object is found at the sampling position which was not found with previous samples, the area is simply subdivided into 4 sub-areas and these areas are pushed back into the queue. Otherwise, the area will also be subdivided as in the previous case but additionally the sub-area that included the sample x will be further subdivided to guide the subdivision into populated areas first.

6.3.2 Exploiting Frame-to-Frame Coherence

Since object distribution rarely changes rapidly from frame to frame we can re-use successful sample positions which found visible objects in a previous frame. The samples are transformed by a global simple translation in image space to approximately compensate the camera movement between two consecutive frames.

The subdivision scheme from Section 6.3.1 will only be initiated after the previously successful samples were tested again. The total number of newly generated samples per frame will be limited to $n^{\frac{2}{3}} - s$, where s is the number of samples of the previous frame to test only a maximum of $n^{\frac{2}{3}}$ samples. Additionally, the number of samples will also be limited by the number of pixels times the (approximated) average projected size in pixels of successful samples in the previous frame.

Figure 6.5 shows an example distribution of image samples in two consecutive frames. At frame 1 no coherence information was used, but most samples were adaptively placed in the

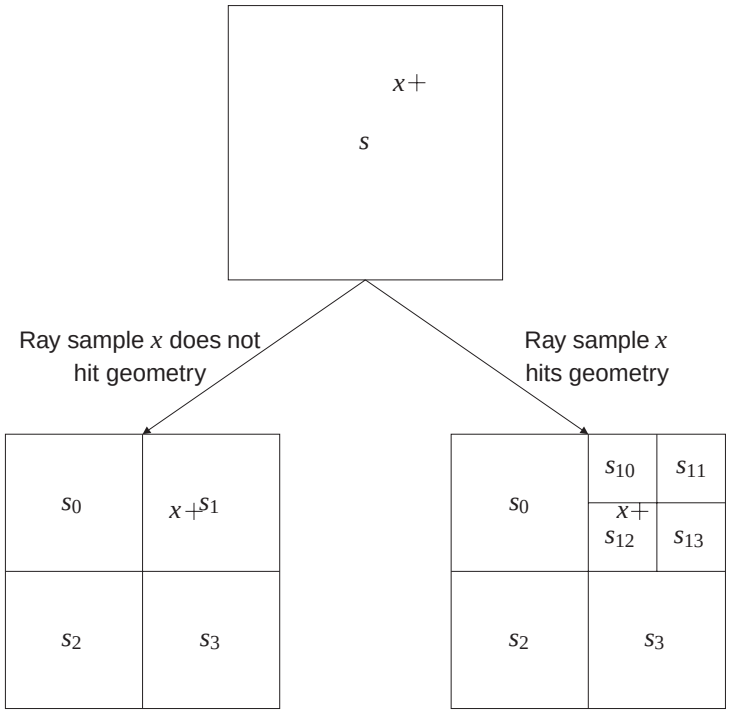


Figure 6.4 Subdivision of sampling areas. If previously untested geometry is found at position x, the sub-division gets finer around x.

densely populated areas of the image. At frame 2 only very few samples miss geometry and many potential occluders are found only by using a very small number of visibility samples.

6.4 Frame Coherent Occlusion Testing Control

In this section we will describe a valuable technique for dynamic control of occlusion culling tests. As we have learned from Section 6.2.3, expensive occlusion tests must be avoided if we can expect a "*potentially visible*" result.

To achieve this, we add an extra *activation* flag to each inner node of the scene geometry. Only if the activation flag is set we will perform the occlusion test. Initially we set the activation flag for every node. The basic idea is to gain visibility results from previous frames by exploiting frame-to-frame coherence. We assume that there is a rather low probability (on average) that an object is occluded in frame i while it is visible in frame $i+1$. Thus, we disable the activation flag if the occlusion test returns *potentially visible*.

6.4.1 Node Activation Oracle

Asymptotically, this algorithm would disable all the activation flags over time. Thus, we need a technique that re-activates scene nodes after several frames. This is done using an *activation oracle* which knows when an occlusion test must be re-initiated. Our current implementation of this oracle re-activates nodes t frames after the de-activation (to compensate for temporary

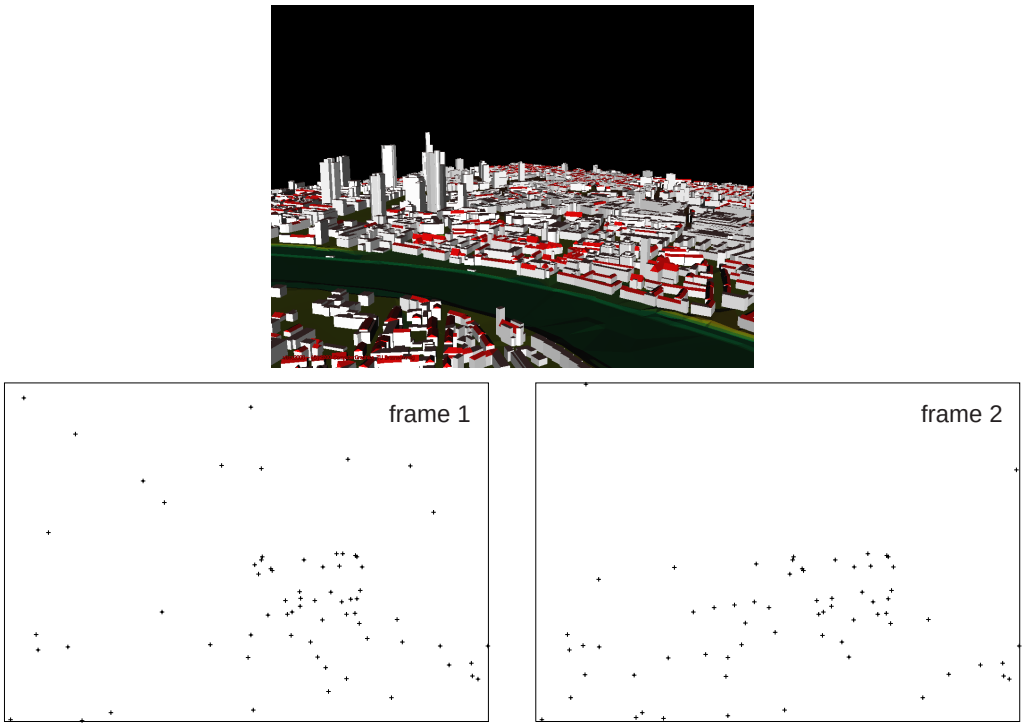


Figure 6.5 Distribution of image sample positions on test scene *Frankfurt*. Frame-to-frame coherence is exploited to find visible object with low computational effort.

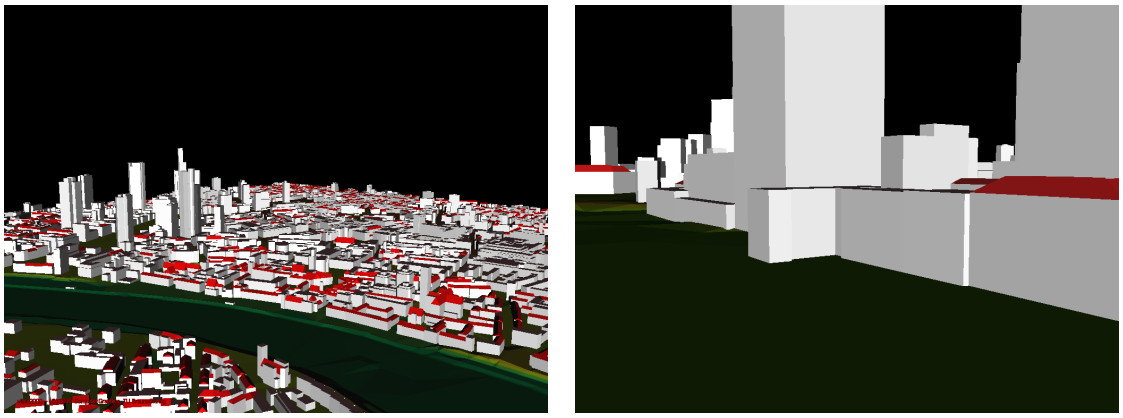


Figure 6.6 Example walkthrough scenarios. Low geometrical occlusion on the left, high on the right.

changes in the view such as moving object in front of complex geometry). Additionally, if no node is re-activated at a certain frame, a random node is selected from the visited nodes and re-activated. If re-activation succeeds (i.e. the occlusion test after re-activation returns "*invisible*)", we also re-activate the node's parent node in the hope to find a larger occluded region.

This occlusion test control is the core of our occlusion culling optimization and can be used in conjunction with many other occlusion algorithms. It is not limited to hierarchical occlusion culling schemes but allows for certain optimization such as the propagation of visibility results up and down the scene hierarchy.

6.5 Results

We have tested our algorithms for occlusion culling control on a 3D city model of Frankfurt consisting of more than 850.000 triangles. Two completely different walkthrough scenarios have been created for testing the algorithm in high- and low-occlusion environments. The low-occlusion scenario is a flight over the city, where the high-occlusion scenario is a walkthrough between houses and skyscrapers at an average human height above ground (Figure 6.6).

On the hardware side, we have used an Intense3D/Wildcat 4210 graphics system that is capable to perform the HP occlusion test [SOG98] in hardware. We have also implemented or used different occlusion test such as [Inc99], but the relative results were quite similar to the Intense3D system, thus we consider the Wildcat a representative environment.

Figures 6.7 to 6.9 compare three culling algorithms. All algorithms traverse the same object hierarchy that was created using the technique presented in Section 6.2.1.

view frustum culling (VFC) implements standard view frustum culling from Section 6.2.2,

occlusion culling (OC) implements classical hierarchical occlusion culling from Section 6.2 including view frustum culling,

adaptive occlusion culling (AOC) implements occlusion culling plus our extensions of Sections 6.3 and 6.4.

The low-occlusion scenario is rather a worst-case situation for every occlusion culling algorithm since there is hardly any occlusion. We can expect that no occlusion culling algorithm will beat a simple VFC here. Surprisingly, our AOC performs very well under this situation. The frame rate curves from Figure 6.7 go hand in hand for VFC and AOC. Figure 6.8 gives us the explanation: the number of occlusion tests performed is very low because of de-activation of occlusion tests. In contrast, OC will test every scene node for occlusion with very low success, only 20% of these test were successful (Fig.6.9), which costs around 25% of rendering performance.

In the high-occlusion scenario, occlusion culling algorithms outperform VFC by an order of magnitude. More interesting is a comparison between OC and AOC. Again, AOC beats OC by around 25%. From Figures 6.8 and 6.9 can we see that the number of occlusion tests performed by AOC is much lower that those of OC. Again, AOC has found the relevant occlusion tests automatically.

From Figure 6.8 we can derive the minimal number of inner nodes that must be traversed when rendering the scene. This number equals the number of occlusion tests for OC since the

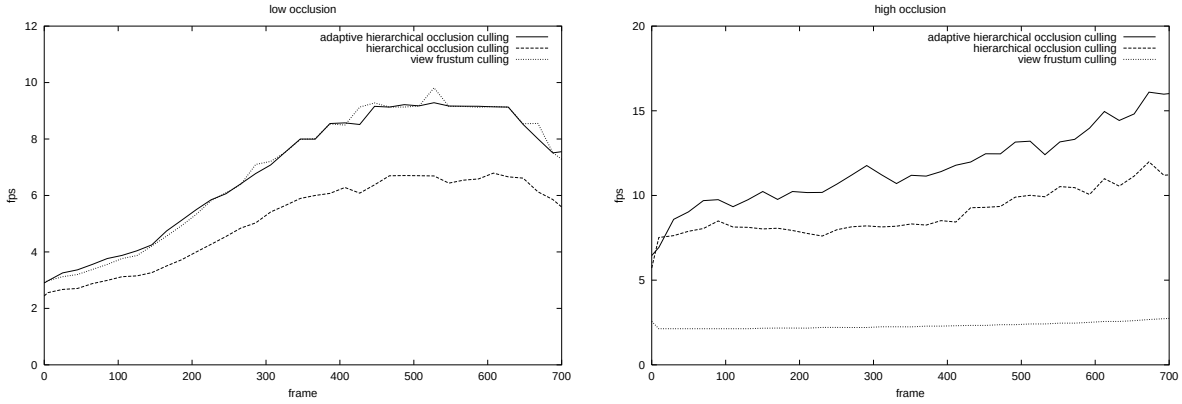


Figure 6.7 Occlusion culling performance.

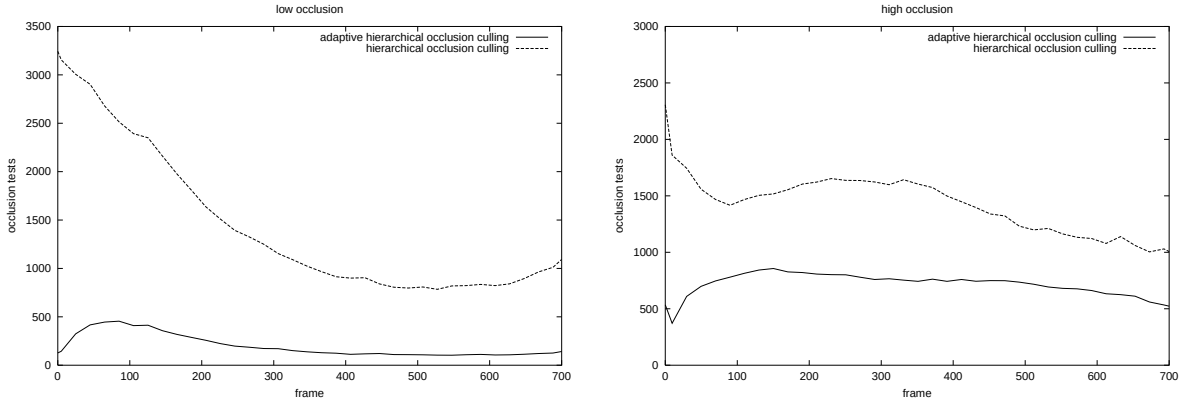


Figure 6.8 Occlusion culling tests.

occlusion test is performed at each inner node. For AOC the fraction of nodes tested for occlusion compared to nodes visited is around 0.1 for the low-occlusion scenario, while it rises up to 0.5 for the high-occlusion scenario. Occlusion tests are activated when it is useful, otherwise they are disabled. This works completely automatically without any user intervention. In both scenarios the number of unsuccessful occlusion tests is very low compared to successful tests. The achievable performance is very competitive to VFC and OC by combining the individual strengths of these methods.

6.6 Conclusions

In this chapter we have presented an innovative occlusion culling scheme that improves hierarchical occlusion culling techniques by avoiding unnecessary visibility tests. Our algorithm does not make any special assumption on the distribution of objects in space such as many occlusion culling schemes for architectural walkthroughs or 2.5D-data sets like height fields [COFHZ98]. Also, it adapts extremely well to the degree of occlusion of a given scene. If all objects are

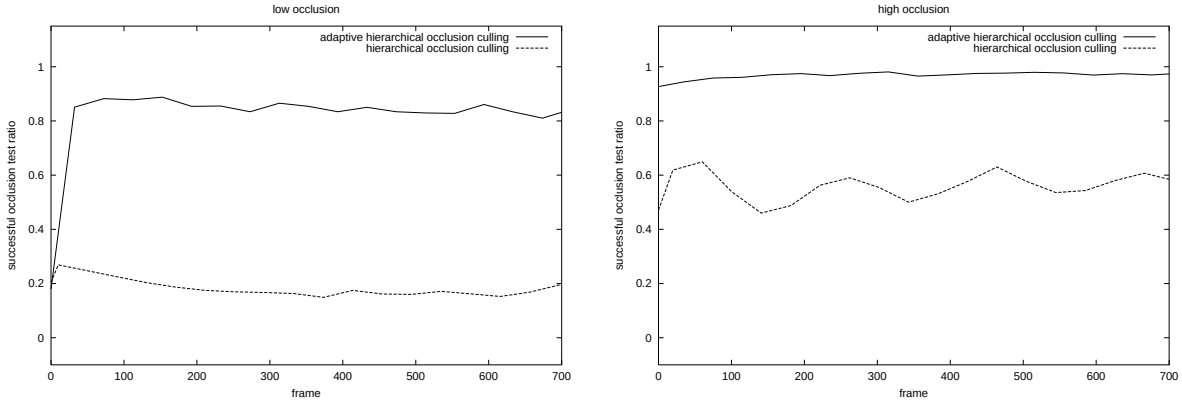


Figure 6.9 Occlusion culling efficiency.

visible and no occlusion occurs, visibility tests are deactivated. In this situation there is almost no overhead compared to raw rendering of such a scene without culling techniques. On the other hand, if many objects are occluded our algorithm not only reduces the total number of occlusion tests, it only identifies the significant nodes for testing.

There is no pre-processing needed – only a hierarchical scene representation must be available which is often the case. Otherwise we can generate a bounding volume hierarchy within a few seconds using the algorithm presented in Section 6.2.1. The algorithm may easily be used to dynamic or at least partially dynamic environments, since we have not made special assumptions that the scene geometry must be static.

In our implementation of the algorithm we used axes-aligned boxes as bounding volumes, since they seem to be a good compromise between tight fitting volumes and fast visibility testing. Alternatively, we could have used *triboxes* [CR99], which promise tighter bounding volumes which might be faster to test for occlusion.

CHAPTER 7

Conclusion

In this thesis we have presented several acceleration schemes for global illumination algorithms and real-time rendering. We have identified object hierarchies as a compact model for scene representation and a natural decomposition scheme for efficient visibility determination. A major contribution of this thesis is the presentation of a novel algorithm for fully automatic generation of object hierarchies using a cost function that minimizes the expected rendering costs for traversing the scene geometry. None of our algorithms makes special assumptions on the type of graphical primitives beeing used (polygons, spheres, etc.), such that all of these primitives may be easily integrated into a system using our algorithms by simply redefining individual object costs used within the cost function.

7.1 Thesis Summary

7.1.1 Ray Acceleration

Making use of the hierarchy generation algorithm, we have developed a new hybrid ray acceleration scheme that combines the advantages of regular grids and bounding volume hierarchies. Based on an available object hierarchy neighbouring nodes are automatically classified to be candidates for insertion into a common grid. This process starts from the leaf of the object hierarchy and propagates hierarchically up to the root. The outcome is an object hierarchy that contains grid nodes and simple bounding volumes nodes referencing sub-scenes. Grid nodes are automatically created in regions that contain homogeneously distributed nearby objects. Thus, a data structure is set up that reflects the appropriate acceleration scheme (grids or object hierarchies) dependent on object distribution and size. We could improve the ray tracing performance using this hybrid approach compared to other implemented state-of-the-art acceleration schemes.

7.1.2 Hierarchical Radiosity

We have also used the ray acceleration scheme successfully for form-factor-calculation in our Radiosity framework. More importantly, the algorithm has been proven to create a hierarchy well suited for radiosity clustering. The resulting bounding volumes are very tight and closeby objects form grids, which can be interpreted as object clusters in the context of hierarchical radiosity algorithms that speed-up the computation of energy transfer between clusters of objects instead of using single objects. We had to invent several techniques that minimize the total

number of tree traversal operations, which are relevant for deep object hierarchies like the ones generated by our hierarchy generation algorithm. We could show that it is feasible to compute accurate radiosity solutions even for highly complex data sets within a few minutes by using object hierarchies.

7.1.3 Occlusion Culling

We applied our hierarchy generation scheme to build the underlying data structure for the hierarchical occlusion culling algorithm. The resulting cull rates (percentage of geometry eliminated by occlusion tests) were higher than the respective rates of alternatively used algorithms in most cases at slightly higher traversal costs as a side effect. Next, we optimized the object traversal code such that most occlusion tests can be avoided. E.g., our rendering algorithm automatically adapts to the degree of occlusion in a walktrough, performing many occlusion tests when it is promising and avoiding tests when useless. This is achieved by implementing an oracle that determines if an occlusion test should be performed when doing scene traversal for every scene node. The current implementation of the oracle extrapolates results from previous frames, exploits frame-to-frame coherence, and carefully samples the view frustum using ray samples to update internal statistical data.

7.2 Future Work

So far, we have not yet applied our hierarchy generation algorithm to dynamic environments or to the field of collision detection. It should be rather easy to integrate our basic construction algorithm into such a framework in a first step but we currently would have to rebuild the whole object hierarchy from scratch if any geometry in an optimized object hierarchy changes. Though, we think that it should be possible to extend our hierarchy generation code to support for (minor) dynamic changes in the scene geometry. Update threads may rebuild the object hierarchy whenever computational resources are available concurrently to the main scene traversal thread.

It would be interesting to research our ray tracing code and our hierarchical radiosity algorithm in a distributed environment. Recently, Wald et al. [WBWS01, WSB01] have presented an interesting paper exploiting fine and coarse level parallelism with modern microprocessors using vector operations and distributing parts of a scene over an Ethernet network. But there is currently little research in the field of distributed radiosity, thus we think that the distribution of object clusters may be worth investigating.

7.3 Acknowledgments

We would like to thank all members of the Institute of Computer Graphics at the Technical University of Braunschweig for their contribution to our rendering platform.

The *aircraft* model was kindly provided by LightWork Design Ltd. Thanks to Simon Gibson for the corrected version of the *atrium* scene. The *vrlab* scene was kindly provided by Fraunhofer Institut für Graphische Datenverarbeitung. We would like to thank T-Mobile International

for providing us with the *Frankfurt* scene model. The *wichmann* scene was kindly provided by 4-e-motions.

Funding of this project by the German Research Foundation (DFG) under grants Fe 431/1-2, Fe 431/4-1, and Fe 431/4-3 is gratefully acknowledged.

Bibliography

[Air90] J.M. Airey. *Increasing Update Rates in the Building Walkthrough System with Automatic Model-Space Subdivision and Potentially Visible Set Calculations*. PhD thesis, Department of Computer Science, University of North Carolina, Chapel-Hill, 1990. 40

[AK89] J. Arvo and D. Kirk. A survey of ray tracing acceleration techniques. In A. S. Glassner, editor, *An Introduction to Ray Tracing*, pages 201–261. Academic Press, London, 1989. 14

[ARB90] J. Airey, J. Rohlf, and F. Brooks. Towards Image Realism with Interactive Update Rates in Complex Virtual Building Environments. In *Proc. of ACM Symposium on Interactive 3D Graphics*, pages 41–5, 1990. 40

[Arv90] J. Arvo. Ray tracing with meta-hierarchies. In *SIGGRAPH Course Notes - Advanced Topics in Ray Tracing*, volume 24, August 1990. 15

[BCG+96] G. Barequet, B. Chazelle, L. Guibas, J. Mitchell, and A. Tal. BOXTREE: A Hierarchical Representation for Surfaces in 3D. In *Proc. of Eurographics*, pages 387–396, 1996. 40

[Bli78] J. Blinn. Simulation of wrinkled surfaces. *Computer Graphics (SIGGRAPH '78)*, 12(3):286–292, August 1978. 2

[BMH99] D. Bartz, M. Meißner, and T. Hüttner. OpenGL-assisted Occlusion Culling of Large Polygonal Models. *Computers & Graphics*, 23(5):667–679, 1999. 40, 53

[BS99] D. Bartz and M. Skalej. VIVENDI - A Virtual Ventricle Endoscopy System for Virtual Medicine. In *Proc. of Symposium on Visualization*, pages 155–166, 1999. 40

[Cat74] E. Catmull. *A Subdivision Algorithm for Computer Display of Curved Surfaces*. PhD thesis, University of Utah, 1974. 2

[CCWG88] Michael Cohen, Shenchang Eric Chen, John R. Wallace, and Donald P. Greenberg. A Progressive Refinement Approach to Fast Radiosity Image Generation. In *Computer Graphics (ACM SIGGRAPH '88 Proceedings)*, volume 22, pages 75–84, August 1988. 5

[CDP95] F. Cazals, G. Drettakis, and C. Puech. Filtering, clustering and hierarchy construction: a new solution for ray-tracing complex scenes. In *Proceedings of Eurographics '95*, pages 371–382, 1995. 15, 24, 32

[CLSS95] Per Henrik Christensen, Dani Lischinski, Eric J. Stollnitz, and David H. Salesin. Clustering for Glossy Global Illumination. Technical Report 95-01-07, Department of Computer Science and Engineering, University of Washington, Seattle, Washington, January 1995. 28

[COFHZ98] D. Cohen-Or, G. Fibich, D. Halperin, and E. Zadicario. Conservative visibility and strong occlusion for viewspace partitioning of densely occluded scenes. In *EUROGRAPHICS 98*, pages 243–253, 1998. 59

[CR99] A. Crosnier and J. R. Rossignac. Tribox bounds for three-dimensional objects. In *Computers & Graphics*, volume 23, pages 429–437, 1999. 10, 11, 60

[CT96] S. Coorg and S. Teller. Temporally Coherent Conservative Visibility. In *Proc. of ACM Symposium on Computational Geometry*, pages 78–87, 1996. 40

[CT97] S. Coorg and S. Teller. Real-Time Occlusion Culling for Models with Large Occluders. In *Proc. of ACM Symposium on Interactive 3D Graphics*, pages 83–90, 1997. 40

[DB97] P.J. Diefenbach and N.I. Badler. Multi-pass pipeline rendering: Realism for dynamic environments. In *Proceedings 1997 Symposium on Interactive 3D Graphics*, pages 59–70, April 1997. 2

[Fel96] Dieter W. Fellner. Extensible image synthesis. In Peter Wisskirchen, editor, *Object-Oriented and Mixed Programming Paradigms*, Focus on Computer Graphics, pages 7–21. Springer, 1996. 25

[FKN80] H. Fuchs, Z. Kedem, and B. Naylor. On Visible Surface Generation by a Priori Tree Structures. In *Proc. of ACM SIGGRAPH*, pages 124–133, 1980. 39, 40

[FTI86] A. Fujimoto, T. Tanaka, and K. Iwata. ARTS: Accelerated ray-tracing system. *IEEE CG&A*, 6(4):16–26, April 1986. 15

[GH96] Simon Gibson and R. J. Hubbold. Efficient hierarchical refinement and clustering for radiosity in complex environments. *Computer Graphics Forum*, 15(5):297–310, December 1996. 28, 33, 35

[GKM93] N. Greene, M. Kass, and G. Miller. Hierarchical Z-Buffer Visibility. In *Proc. of ACM SIGGRAPH*, pages 231–238, 1993. 7, 40, 53

[Gla84] Andrew S. Glassner. Space subdivision for fast ray tracing. *IEEE CG&A*, 4(10):15–22, October 1984. 13, 15

[Gla89] Andrew S. Glassner. *Introduction to Ray Tracing*. Academic Press, New York, NY, 1989. 5

[GLM96] S. Gottschalk, M. Lin, and D. Manocha. OBBTree: A Hierarchical Structure for Rapid Interference Detection. In *Proc. of ACM SIGGRAPH*, pages 171–180, 1996. 12, 40

[Gre95] N. Greene. *Hierarchical Rendering of Complex Environments*. PhD thesis, Computer and Information Science, University of California, Santa Cruz, 1995. 40

[Gre96] N. Greene. Hierarchical Polygon Tiling with Coverage Masks. In *Proc. of ACM SIGGRAPH*, pages 65–74, 1996. 40

[Grö95] Alwin Gröne. *Entwurf eines objektorientierten Visualisierungssystems auf der Basis von Raytracing*. Ph. D. thesis, Fakultät für Informatik der Eberhard-Karls-Universität Tübingen, 1995. In German. 12, 21, 29

[GS87] J. Goldsmith and J. Salmon. Automatic creation of object hierarchies for ray tracing. *IEEE Computer Graphics and Applications*, 7(5):14–20, May 1987. 7, 12, 14, 16, 17, 26, 29

[GTGB84] Cindy M. Goral, Kenneth E. Torrance, Donald P. Greenberg, and Bennett Battaile. Modelling the Interaction of Light Between Diffuse Surfaces. In *Computer Graphics (ACM SIGGRAPH '84 Proceedings)*, volume 18, pages 212–222, July 1984. 5

[Hai87] E. A. Haines. A proposal for standard graphics environments. *IEEE Computer Graphics and Applications*, 7(11):3–5, November 1987. code available via *ftp://princeton.edu/pub/Graphics*. 22

[HDSD99] J. M. Hasenfratz, C. Damez, F. Sillion, and G. Drettakis. A practical analysis of clustering strategies for hierarchical radiosity. *Computer Graphics Forum (Proc. Eurographics '99)*, 18(3):C–221–C–232, September 1999. 7, 28, 36

[HKM96] M. Held, J.T. Klsowski, and J.S.B. Mitchell. Real-time collision detection for motion simulation within complex environments. *Visual Proceedings (SIGGRAPH'96)*, 1996. 12

[HMC+97] T Hudson, D. Manocha, J. Cohen, M. Lin, Kenneth E. Hoff, and H. Zhang. Accelerated Occlusion Culling Using Shadow Frusta. In *Proc. of ACM Symposium on Computational Geometry*, pages 2–10, 1997. 40

[HMK+97] L. Hong, S. Muraki, A. Kaufman, D. Bartz, and T. He. Virtual Voyage: Interactive Navigation in the Human Colon. In *Proc. of ACM SIGGRAPH*, pages 27–34, 1997. 39

[Hop96] Hugues Hoppe. Progressive Meshes. In *Computer Graphics Proceedings, Annual Conference Series, 1996 (ACM SIGGRAPH '96 Proceedings)*, pages 99–108, 1996. Available from http://www.research.microsoft.com/research/graphics/hoppe. 39, 51

[HSA91] Pat Hanrahan, David Salzman, and Larry Aupperle. A Rapid Hierarchical Radiosity Algorithm. In *Computer Graphics (ACM SIGGRAPH '91 Proceedings)*, volume 25, pages 197–206, July 1991. 6, 33, 35

[Inc99] Silicon Graphics Inc. *Silicon Graphics 320 Visual Workstation*. 1999. available from http://visual.sgi.com/products/320/index.html. 58

[Jan86] F. W. Jansen. Data structures for ray tracing. In L. R. A. Kessener, F. J. Peters, and M. L. P. van Lierop, editors, *Data Structures for Raster Graphics*, Eurographic Seminars, pages 57–73. Springer, 1986. 13, 15

[Kaj86] James T. Kajiya. The Rendering Equation. In *Computer Graphics (ACM SIGGRAPH '86 Proceedings)*, volume 20, pages 143–150, August 1986. 4

[KHM+98] J.T. Klosowski, M. Held, J.S.B. Mitchell, H. Sowizral, and K. Zikan. Efficient collision detection using bounding volume hierarchies of k-dops. *IEEE Transactions on Visualization and Computer Graphics*, 4(1), 1998. 12

[KK86a] T. L. Kay and J. T. Kajiya. Ray tracing complex scenes. *Computer Graphics (ACM SIGGRAPH '86 Proceedings)*, 20(4):269–278, 1986. 12, 14, 16, 17, 29

[KK86b] T. L. Kay and J. T. Kajiya. Ray Tracing Complex Scenes. In *Proc. of ACM SIGGRAPH*, pages 269–278, 1986. 42

[KMM+98] J. Klosowski, M.Held, J. Mitchell, H. Sowizral, and K. Zikan. Efficient Collision Detection Using Bounding Volume Hierarchies of k-DOPs. *IEEE Transactions on Visualization and Computer Graphics*, 4(1):21–36, 1998. 40

[KS97] K. Klimansezewski and T. Sederberg. Faster ray tracing using adaptive grids. *IEEE Computer Graphics and Applications*, 17(1):42–51, 1997. 26, 32

[LG95] D. Luebke and C. Georges. Portals and Mirrors: Simple, Fast Evaluation of Potentially Visible Sets. In *Proc. of ACM Symposium on Interactive 3D Graphics*, pages 105–106, 1995. 40

[LKM01] Erik Lindholm, Mark Kilgard, and Henry Moreton. A user programmable vertex engine. 2001. http://developer.nvidia.com/view.asp?IO=SIGGRAPH_2001. 2

[MF99] G. Müller and D. Fellner. Hybrid Scene Structuring with Application to Ray Tracing. In *Proceedings of the International Conference on Visual Computing (ICVC'99)*, pages 19–26, Goa, India, 1999. available online via http://www.cg.cs.tu-bs.de/people/mueller/publications. 30, 32, 33, 36, 42, 51

[MH99] T. Möller and E. Haines. *Real-Time Rendering*. A.K. Peters, 1999. 10

[MSF99] Gordon Mueller, Stephan Schaefer, and Dieter Fellner. Automatic creation of object hierarchies for radiosity clustering. In *Proceedings of Pacific Graphics '99 (Seventh Pacific Conference on Computer Graphics and Applications)*, Los Alamitos, CA, October 1999. IEEE Computer Society Press. 51, 52

[Mül97] G. Müller. Beschleunigung strahlbasierter rendering algorithmen. M.Sc.. thesis, Rheinische Friedrich-Wilhelms-Universität Bonn, May 1997. in German, available online via http://www.cg.cs.tu-bs.de/people/mueller/. 15, 23

[Nay92] B. Naylor. Partitioning Tree Image Representation and Generation From 3D Geometric Models. In *Proc. of Graphics Interface*, pages 201–212, 1992. 40

[NN85] Tomoyuki Nishita and Eihachiro Nakamae. Continuous Tone Representation of Three-Dimensional Objects Taking Account of Shadows and Interreflection. In *Computer Graphics (ACM SIGGRAPH '85 Proceedings)*, volume 19, pages 23–30, July 1985. 5

[NVI01] NVIDIA. Per pixel lighting in opengl. 2001. http://developer.nvidia.com/-view.asp?IO=gdc2001_perpixellighting. 3

[Pho75] B.-T. Phong. Illumination for computer generated pictures. *Communications of the ACM*, 18(6):311–317, June 1975. 2

[Rog97] D.F. Rogers. *Procedural Elements for Computer Graphics, 2ed.* McGraw Hill, New York, 1997. 1

[RTL84] R.L.Cook, T.Porter, and L.Carpenter. Distributed ray tracing. volume 18, pages 137–145, July 1984. 4

[SAG94] Brian Smits, James Arvo, and Donald Greenberg. A Clustering Algorithm for Radiosity in Complex Environments. In *Computer Graphics Proceedings, Annual Conference Series, 1994 (ACM SIGGRAPH '94 Proceedings)*, pages 435–442, 1994. 6, 7, 27, 34, 35

[Sam94] H. Samet. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, Reading, 1994. 39

[Sch97] G. Schaufler. Nailboards: A rendering primitive for image caching in dynamic scenes. In *Rendering Techniques '97 (Proceedings of the Eighth Eurographics Workshop on Rendering)*, pages 151–162, 1997. 51

[Sch00] S. Schäfer. *Efficient Object-Based Hierarchical Radiosity Methods*. Ph. D. thesis, Institut für ComputerGraphik, Technische Universität Braunschweig, 2000. 5, 6

[SG96] O. Sudarsky and C. Gotsman. Output-Sensitive Visibility Algorithms for Dynamic Scenes with Applications to Virtual Reality. In *Proc. of Eurographics*, pages 249–258, 1996. 40

[SGI97] SGI. *OpenGL Optimizer Manual*. Silicon Graphics Inc., Mountain View, 1997. 40

[Sil95] Francois Sillion. A Unified Hierarchical Algorithm for Global Illumination with Scattering Volumes and Object Clusters. *IEEE Transactions on Visualization and Computer Graphics*, 1(3), September 1995. 6, 7, 27, 34, 35

[SL98] J. Snyder and J. Lengyel. Visibility Sorting and Compositing without Splitting for Image Layer Decompositions. In *Proc. of ACM SIGGRAPH*, pages 219–231, 1998. 40

[SOG98] N. Scott, D. Olsen, and E. Gannett. An Overview of the VISUALIZE fx Graphics Accelerator Hardware. *The Hewlett-Packard Journal*, (May):28–34, 1998. 40, 53, 58

[SP94] Francois Sillion and Claude Puech. *Radiosity and Global Illumination*. Morgan Kaufmann, San Francisco, CA, 1994. 35

[Spa90] J. N. Spackman. *Scene Decompositions for Accelerated Ray Tracing*. PhD thesis, England, February 1990. 21

[SS92] K. Sung and P. Shirley. Ray tracing with the bsp tree. In D. Kirk, editor, *Graphic Gems III*, pages 271–274. Academic Press, San Diego, 1992. 23

[SS96] G. Schaufler and W. Stürzlinger. A three dimensional image cache for virtual reality. In *Computer Graphics Forum (Proc. Eurographics '96)*, pages 227–236, 1996. 51

[SSS98] Marc Stamminger, Philipp Slusallek, and Hans-Peter Seidel. Bounded clustering: Finding good bounds on clustered light transport. In *Pacific Graphics '98*, Singapore, October 1998. 28

[TS91] S. Teller and C.H. Sequin. Visibility Pre-processing for Interactive Walkthroughs. In *Proc. of ACM SIGGRAPH*, pages 61–69, 1991. 40, 51

[WBWS01] Ingo Wald, Carsten Benthin, Markus Wagner, and Philipp Slusallek. Interactive rendering with coherent ray tracing. In Alan Chalmers and Theresa-Marie Rhyne, editors, *Computer Graphics Forum (Proceedings of EUROGRAPHICS 2001)*, volume 20, pages 153–164. Blackwell Publishers, Oxford, 2001. available at http://graphics.cs.uni-sb.de/ wald/Publications. 62

[Whi80] Turner Whitted. An improved illumination model for shaded display. *Communications of the ACM*, 23(6):343–349, June 1980. 4, 14

[WSB01] Ingo Wald, Philipp Slusallek, and Carsten Benthin. Interactive distributed ray tracing of highly complex models. In S.J.Gortler and K.Myszkowski, editors, *Rendering Techniques 2001 (Proceedings of the 12th EUROGRAPHICS Workshop on Rendering)*, pages 277–288. Springer, 2001. available at http://graphics.cs.uni-sb.de/ wald/Publications. 62

[WWS01] P. Wonka, M. Wimmer, and F.X. Sillion. Instant visibility. *EUROGRAPHICS 2001*, 20(3), 2001. 51

[ZMHH97] H. Zhang, D. Manocha, T. Hudson, and Kenneth E. Hoff. Visibility Culling Using Hierarchical Occlusion Maps. In *Proc. of ACM SIGGRAPH*, pages 77–88, 1997. 39, 40, 53

[ZSD+00] D. Zorin, P. Schröder, T. DeRose, L. Kobbelt, A. Levin, and W. Sweldens. Subdivision for Modeling and Animation. In *ACM SIGGRAPH Course 23*, 2000. 39

Publications

Journals and Books

D. W. FELLNER, S. HAVEMANN, G. MÜLLER: Modeling of and Navigation in Complex 3D Documents. *Computers & Graphics*, Vol. 22, No. 6, pp. 645–653, December 1998.

G. MÜLLER, S. SCHÄFER, D.W. FELLNER: Automatic Creation of Object Hierarchies for Radiosity Clustering. *Computer Graphics forum*, Vol. 19, No.4, number 4, pp. 213-221, December 2000.

D.W. FELLNER, J. HABER, S. HAVEMANN, L. KOBBELT, H.P.A. LENSCH, G. MÜLLER, I. PETER, R. SCHNEIDER, H.-P. SEIDEL, W. STRASSER: Beiträge der Computergraphik zur Realisierung eines verallgemeinerten Dokumentbegriffs, *Informationstechnik und Technische Informatik (it+ti)*, Vol. 42, No. 6, pp. 8–16, Oldenbourg Verlag, 2000. (in German)

D. BARTZ, M. MEISSNER, G. MÜLLER Efficient Occlusion Culling for Large Model Visualization, In Frits Post, Georges-Pierre Bonneau, and Gregory M. Nielson (Editors), *Scientific Visualization (Dagstuhl 2000)*, 2000.

G. MÜLLER, D.W. FELLNER: An Adaptive Hierarchical Occlusion Culling Algorithm for Interactive Large Model Visualization, *Computer Graphics forum*, 2001. (submitted)

Conference Proceedings and Technical Reports

G. MÜLLER, D.W. FELLNER: Hybrid Scene Structuring with Application to Ray Tracing. *Proceedings of International Conference on Visual Computing (ICVC'99)*, pp. 19–26, Goa, India, February 1999.

M. MEISSNER, D. BARTZ, T. HÜTTNER, G. MÜLLER, J. EINIGHAMMER: Generation of Decomposition Hierarchies for Efficient Occlusion Culling of Large Polygonal Models, WSI/GRIS technical report, University of Tübingen, WSI-99-13, ISSN 0946-3852, June 1999.

G. MÜLLER., S. SCHÄFER , D.W. FELLNER: A Rapid Clustering Algorithm for Efficient Rendering. *Short Papers and Demos (Eurographics '99)*, 1999.

G. MÜLLER, S. SCHÄFER, D.W. FELLNER: Automatic Creation of Object Hierarchies for

Radiosity Clustering. *Proceedings of Pacific Graphics '99 (Seventh Pacific Conference on Computer Graphics and Applications)*, IEEE Computer Society Press, Seoul, Korea, October 1999.

M. MEISSNER, D. BARTZ, T. HÜTTNER, G. MÜLLER, J. EINIGHAMMER: Generation of Decomposition Hierarchies for Efficient Occlusion Culling of Large Polygonal Models, *Vision, Modeling, and Visualization (VMV 2001)*, Stuttgart, November 2001.